H446 A level NEA

Centre: Beacon Academy

Centre Number: 56320

Candidate Number:

Name: Alfred Buckingham

Title: Secure Password Generator and Manager

# Table of Contents

Contents

[Table of Contents 2](#_Toc165015494)

[Analysis 4](#_Toc165015495)

[Outline 4](#_Toc165015496)

[Stake Holders 4](#_Toc165015497)

[Research 5](#_Toc165015498)

[Inspiration 5](#_Toc165015499)

[Stake holder input 6](#_Toc165015500)

[Computational methods 8](#_Toc165015501)

[Features 8](#_Toc165015502)

[Scope 9](#_Toc165015503)

[Hardware and Software Requirements 10](#_Toc165015504)

[Success Criteria 11](#_Toc165015505)

[Design 13](#_Toc165015506)

[Explanation 13](#_Toc165015507)

[Key variables and data structures 14](#_Toc165015508)

[Test Data for Development 16](#_Toc165015509)

[Post development testing 18](#_Toc165015510)

[Sketch of screen layout 21](#_Toc165015511)

[Key Algorithms and Framework 23](#_Toc165015512)

[Developing the Coded Solution 28](#_Toc165015513)

[Development checklist 28](#_Toc165015514)

[Milestone 1: Cypher Algorithm 28](#_Toc165015515)

[Milestone 2: Random Password Generation 32](#_Toc165015516)

[Milestone 3: Password Strength Checker 35](#_Toc165015517)

[Milestone 4: account name and master password storage 41](#_Toc165015518)

[Milestone 5: Interface 43](#_Toc165015519)

[Milestone 6: importing the algorithms. 46](#_Toc165015520)

[Milestone 7: Encryption and Saving 51](#_Toc165015521)

[Testing to inform evaluation. 55](#_Toc165015522)

[60](#_Toc165015523)

[60](#_Toc165015524)

[61](#_Toc165015525)

[61](#_Toc165015526)

[Evaluation 62](#_Toc165015527)

[Stake Holder Input for Evaluation 62](#_Toc165015528)

[What I Could Improve 63](#_Toc165015529)

[What I have learned 64](#_Toc165015530)

[Bibliography 64](#_Toc165015531)

[Appendices 64](#_Toc165015532)

[Code dump 64](#_Toc165015533)

[**Encryption and decryption** 64](#_Toc165015534)

[**main menu interface** 66](#_Toc165015535)

[**master password storage** 72](#_Toc165015536)

[**Password generation** 77](#_Toc165015537)

[**Save to file** 79](#_Toc165015538)

[**Password strength checker** 80](#_Toc165015539)

# 

# Analysis

## Outline

For this project I will make an encrypted password generator with a way to store them in a secure file. I will use the tkinter python module for the graphical interface and integrate code for a randomised password generator and a way to encrypt it. I will then use an external program to store the passwords in a table with the name of the websites and the encrypted passwords in a neat way. I could store the password and website names in an excel spreadsheet or I could store it in a list in a notepad file which is locked by password access. The password will be displayed encrypted where it can be pasted into the program to decrypt it ready to paste into the log in box. I can encrypt the original so that it is hidden, and it will be decrypted upon copying the text in the program. This can be then further expanded to be an account management system in general. Inspiration for this project is applications such as Dashlane, NordPass, and Keeper.

## Stake Holders

My primary stake holders will be Steven B, Adil K, and Dominic S. I have frequent contact with these people. Steven is a disorganised person who creates strong passwords but struggles to remember them and often finds himself resetting his passwords which wastes a lot of time. Dominic struggles to remember different passwords so uses repeat passwords which is less secure. Adil also uses repeating passwords for various sites. This solution offers much more security being encrypted randomised passwords. It will be a simple intuitive desktop app that can be used to quickly find the encrypted passwords. As well as this it provides a much simpler way to remember passwords as the only password, they need to remember is the one to access the application and the locked folder.

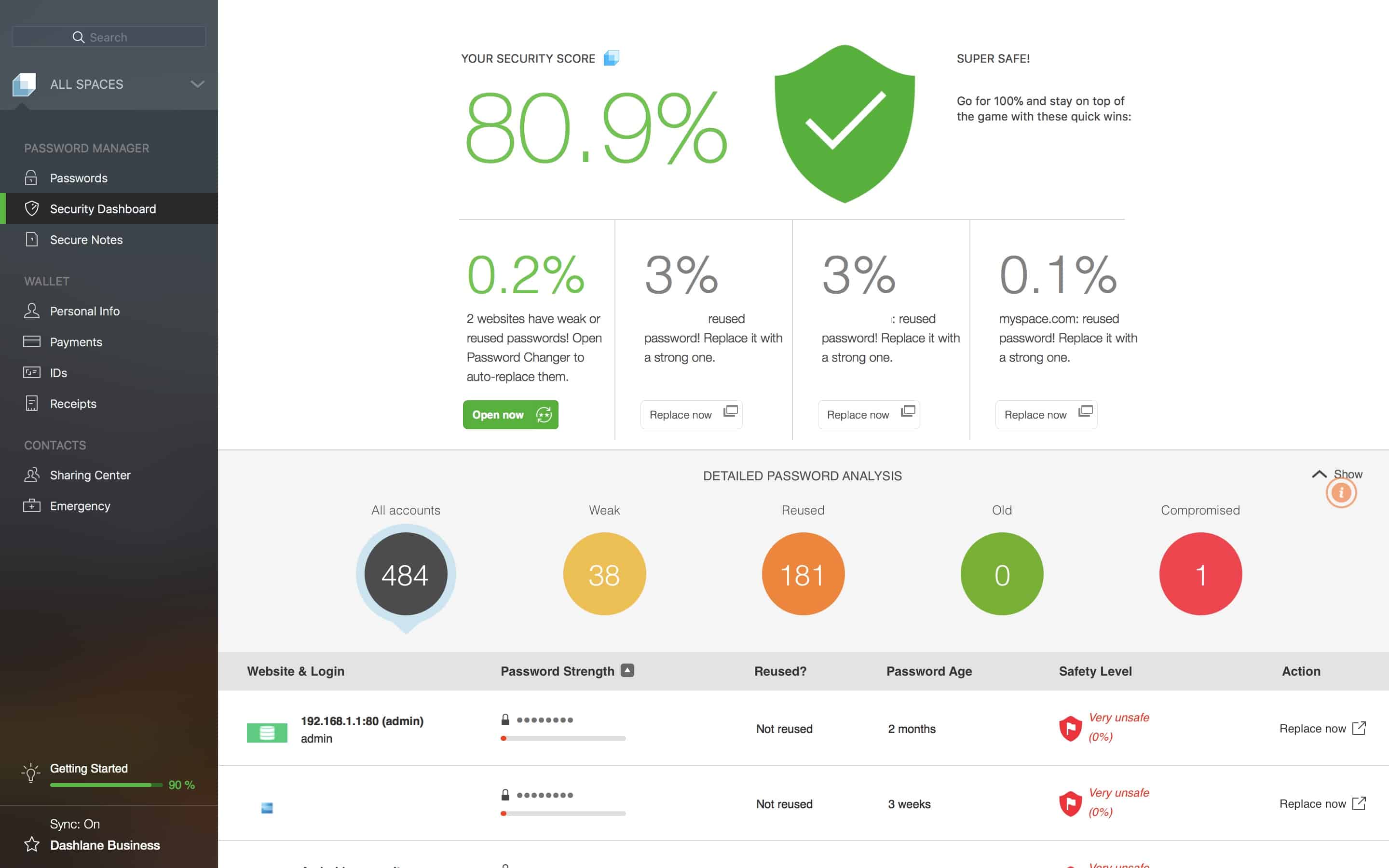
## Research

### Inspiration

* Dashlane ([Link to Dashlane](https://www.dashlane.com/business-password-manager/try?utm_source=adwords&utm_campaign=UK_Search_Brand_Exact_MaxConv&utm_medium=sitelink&utm_term=RSA3_August2022&gclid=EAIaIQobChMI0tKbrsjM_gIVQ7rVCh1smwiZEAAYASAAEgJCd_D_BwE))



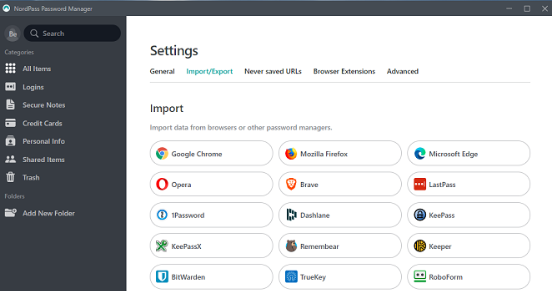
Dashlane is a paid service that is integrated to chrome web browsers. It creates and stores random passwords and enables the user to fill out account details with one click after providing a single password for the service. As well as this Dashlane provides other security measures such as checking the strength of various passwords. The most notable feature of Dashlane is the way it is integrated into google which makes it much easier to use even when the app is not open. Dashlane has a lot of functionality and prioritises functionality over the user interface which seems cluttered at first glance.



* NordPass ([link to NordPass](https://nordpass.com/?gad=1&gclid=EAIaIQobChMI0Z-VntvY_gIVCbLtCh2JawOYEAAYASAAEgLu_fD_BwE))



NordPass is an extension to the popular VPN service Nord VPN. NordPass similar to Dashlane is a desktop app that can be integrated into any chrome based web browser. It auto-saves and automatically fills out information boxes that it has saved data for in the database in the free version. The paid service has added security features such as password strength checks, scans for repeated passwords and breach checks. The main feature that makes NordPass stand out is that it can import the data of other password managers when given the correct permissions. This means that it can quickly pull all of the data across the various accounts and store it all into one place. NordPass also has a very simplistic user interface with a modern style which makes it easy to navigate and understand the app.



### Stake holder input

I will ask the same 4 questions to each stake holder in an attempt to find a common issue and some key features that will be required to fulfil their needs. Each question will have the aim of either finding problems with account and password management and finding ways to improve it. As well as this they can also help to find further wanted features.

Question 1) “what is the main issue you have with managing your accounts and passwords?” – this question will be key to finding the solution to the correct problem in the project.

Question 2) “what additional features would make managing your accounts and passwords easier?” – this will help to find useful additional features.

Question 3) “what kind of interface would you prefer to see for this kind of program?” – this will help to find the preferred style of application whether it shows more functionality or if it is simpler to use

Question 4) “what is your main goal with a program like this?” – this will help to find the priorities of the stake holders with an account management and password system.

Adil K- This stakeholder wants to use the program to assist with storing and remembering the passwords he uses. He also wants other members of his family to be able to use the program.

1) “I normally forget which emails I use and which passwords I use” – given this feedback, I will need to find a way to pair the account name to the password in the database or include a forgot password button.

2) “Have multiple users able to access the same application with different logins like if a different family member wants to log in to it with their account” – this means the user could have multiple people using the same app therefore requiring some kind of password lock for each user.

3) “I prefer programs that are easy to use” – this shows that I need a more simplified user interface that is easy to use and pick up

4) “I just want to make manging my passwords easier so I can have more varied ones” – this reinforces the fact that the program should be easy to use.

Dominic S- this stakeholder wants to use the program to assist with password creation and security testing. This user will use the program occasionally if he ever forgets a password but intends to use it for domestic purposes.

1)“I keep using the same password because I won’t be able to remember many different ones.” – this shows that a strength checker and a randomised password generator would be important.

2) “it would be cool to have a password strength tester.” – this could be implemented as an additional feature by building a list out of the password and checking if the characters fill out criteria to determine password strength.

3) “apps that fit functionality into a simple UI are the best in my opinion” – this is another stakeholder that wants a more simplified user interface which means I could include multiple systems into one function.

4) “I want my passwords to be more secure” – security testing should be a high priority for this program

Steven B- This stakeholder wants to streamline the process of filling out security fields using this program. He will use this regularly for work and home use alike.

1) “I find it inconvenient to keep finding where my passwords are safely stored” – the method of storing the account details needs to be neatly set out and easy to access for the user

2) “it would be nice if the program could automatically fill out the account details fields” – this would require finding a way to get python to link to chrome so it could recognise where these fields are and fill them in

3) “a good user interface is always helpful.”

4) “I just want to make the whole process of filling out and finding my passwords easier and faster.” – organised storage and potentially automatically filling out account detail fields would help to solve this query

## Computational methods

Procedural thinking – procedural thinking is similar to the procedural programming paradigm. This is where the instructions in the algorithm are split into subroutines that it works through in a specific given order. This ensures that the same process is being carried out each time the program is run. Procedural thinking will be a part of my program when it comes to the password generation and storing it in a database. It will be used because I need to make sure that is using the same process to create the password each time which ensures it will be strong and suitable for the user. As well as this procedural thinking will be used to store all of the data about the accounts in a database. I will also need to have the separate parts of the program such as the database, the password generator, the user interface, and the password strength test.

Thinking ahead – thinking ahead is where certain aspects of the algorithm are completed before others to make it more efficient and save time. Algorithms that use thinking ahead take note of any inputs or conditions that need to be predetermined. This will appear in my password management system in the form of inputs such as the account name and the length of password. There will also need to be predetermined Boolean values in the algorithm that help it to determine where bits of data are stored in the database.

Logical thinking – Logical thinking is where an algorithm must make a branching decision based on a set of variables. My project will require logical thinking components when it comes to generating a password. It will need to choose what characters to apply to each individual list before putting it into a string. The program will also need to decide which location the data will be stored at, especially for each user.

## Features

I have obtained a lot of ideas for features to include in my program through my research and the input of my stakeholders that could amplify the usefulness and accessibility of the program.

The first and main feature in my program will be the login details database. This will likely be in the form of a spreadsheet or note file with a list of all the account usernames, emails and the hashed passwords. This is the central system that my program will use to store the account details and each item will be added to this spreadsheet automatically from the program. This will be stored in an encrypted password protected file and the passwords stored within will be further hashed. In order to access the password, the user will need to paste the hashed version of the password into a “decrypt” box that will have a copy button next to it. This copy button will decrypt the password and save it to the windows clipboard to be pasted into a password field.

In order to add items to the CSV file the user will have to fill out 3 different boxes. Those boxes will be: “website name”, “username/ email”, and “password”. The user will then have to click the “add to database” button. This will fill each field in a table with the text input in the boxes, however for the password box, the program will use a hashing algorithm to encrypt the password before storing it in the CSV file. In a table with 3 columns for the website name, username and the hashed password.

Another major feature of my program will be the random password generator. This will have a very simplistic design with a simplistic interface with one input box for the desired length of the password the program will then use an algorithm using various lists to build a completely randomised password that will always generate a password that will get a strong rating in the built in password strength checker.

The password strength checker is a feature that I found many of the other programs had during my research. I think along with hashing the stored passwords, this is another effective security measure I can realistically include in my program. This will involve an input box with a “check” button. This will then track through each character in the password filling out a checklist of set requirements such as, minimum of 8 characters, capital letters, special characters, and numbers. It will then output a rating of weak, strong or very strong and then update the CSV file with the current password rating.

## Scope

There are some limitations with the features I can include in this type of program that some of the applications in my research used. This could be for various reasons such as not having the required hardware, not having enough time or not having the skill or knowledge to implement some features.

One of the features that was requested by a stakeholder was having the security fields on websites filled out automatically. This feature would be too difficult to achieve given the timescale and requiring programmes that are not currently available to me. To rectify this issue, I could create a copy button which automatically saves the text to the clipboard so it can be pasted directly into the security fields to save time typing the password out.

Another feature I found in my research was a breach detector which could notify the user when a password has been breached and they need to update it. This feature is far too difficult to implement given that I neither have the technology or a large enough database network that will be able to detect when a device with a different IP address accesses the users account. Due to this limiting factor my main source of security will be making sure that the passwords are encrypted when they are stored. I could then decrypt the stored passwords when the user supplies the program with a security pin so it can be viewed and copied.

Storing the passwords across multiple devices will also be a feature I won’t be able to include in this program. This is due to the fact that I don’t have access to a server network that can communicate with multiple devices. Storage across multiple devices could be possible if the user saves the application and the storage files to the cloud. However, this will be user dependent since not every user will have access to cloud storage.

I will be limited in how the user interface looks. I do not have the art skill to create professional looking icons and will be limited not having a dedicated team of professionals to work on the user interface. I will be able to import a few graphical features from free assets on the internet for features like button icons and decoration images.

## Hardware and Software Requirements

Developer Requirements-

Hardware required.

|  |  |
| --- | --- |
| Hardware | Justification |
| 4 GB RAM | This is the minimum amount of RAM required to run IDLE (the IDE I will be using for this project) as well as multitask. |
| 100 Kb of free storage | Being a small program, I do not predict this taking up much space at all so a minimal amount of storage will be required. |
| 8th gen intel I3 or equivalent CPU | It is important to have a CPU that is powerful enough and has enough cores to run each part of the program at the same time |
| Radeon RX 580 or equivalent graphics card | The program will have a graphical interface, so an entry level graphics card is required to test the program |

Software required.

|  |  |
| --- | --- |
| Software | Justification |
| IDLE (Python 3.6- 64 bit) | This will the main IDE I will use to create the program which means it will be potentially the most important piece of software on the project. |
| Windows 10/11 | Even though python can be used on a Windows 7 or Linux. I intend on using Microsoft programs like notepad and excel to store files like the saved account names and the hashed master password. |
| Microsoft Notepad | I want to use this to store individual strings such as the account names and hashed passwords. |
| Microsoft Excel | I will use this to create a spread sheet of the account names websites and passwords |
| Tkinter and pickle libraries for python | The program will need to use these python libraries in order to create the user interface and save the passwords to external files |

User Requirements-

Hardware required.

|  |  |
| --- | --- |
| Hardware | Justification |
| 4 GB of RAM | The minimum amount of RAM required to run python programs from IDLE without affecting the computers multitasking capabilities. |
| 100 Kb of free storage | This is a small program and will not require much space to run |
| 8th gen intel I3 CPU or equivalent | Making use of multiple graphical interfaces at the same time the CPU will need to have enough cores to handle the multitasking required to have this program run at the same time as a web browser |
| Radeon RX 580 or equivalent GPU | The program will require an entry level graphics card in order to run the graphical interfaces |

Software required.

|  |  |
| --- | --- |
| Software | Justification |
| Windows 10/11 | Like the developer, the user would need to have access to both to IDLE and the modern Microsoft apps |
| IDLE | Having IDLE will be required to open the program until I can make it into an executable file |
| Excel | Will be required to see how the passwords are saved once encrypted |
| Tkinter and pickle libraries for python | The program will need to use these python libraries in order to create the user interface and save the passwords to external files |

## Success Criteria

|  |  |  |
| --- | --- | --- |
| Criteria | Measurement/ Evidence | Justification |
| Storing multiple passwords | The user should be able to store at least 3 passwords in a clear locked file. | If the user cannot store the passwords somewhere then there will be nothing to copy into the account details boxes on websites and apps. |
| Strong password randomiser button | The user should be able to generate at least 5 12 character passwords that would get a “Strong” rating when using the built in password checker. | This was one of the main features suggested by stakeholders who struggle with creating strong passwords and working password randomiser |
| Encrypting and decrypting stored passwords | The user should be able to store 3 encrypted passwords in a locked file that do not resemble the original passwords. All 3 should then be successfully decrypted when the copy button is used. | This feature is key to the working of the program. If the user is able to store at least 3 passwords, they will be able to access at least 3 different websites easily. It would also triple the security for those that only use one password for everything |
| Working copy button | The user presses the copy button which copies the encrypted password and decrypts it and save the decrypted text to the clipboard. 3 different passwords should be successfully copied and pasted to see if it matches the original password. | A copy button would save a lot of time tabbing between the stored password and the log in tab for the website. Having one button would enable the user to use that and just paste the text that was saved to the clipboard into the box. |
| Multiple accounts management | The user should be able to create at least 2 accounts that have 2 different databases of stored passwords and account details. | Having different accounts with stored data was one of the requests of the stakeholders. It allows different users to access the same program and store different passwords to protect privacy. |
| The program opens successfully | The user should be able to click on the executable file which opens up to the log in screen. | This would make it so that the user would not need to interact with python in any way in order to use the program. |
| Working password strength checker button | The password strength check should give a valid result for at least 5 passwords (valid result: a result that matches to the checklist of what makes a strong password). | This feature would help keep the user more secure by supplying them with a check to make sure that their passwords are as strong and secure as possible |
| Log in boxes work | The user should be able to type in the given details correctly to log into their specific account | The user needs to be able to log in to the program to access key features that need to be kept secure. |
| Clear and concise design that is easy to interpret for all users | The user should be able to access all features of the program without given |  |

# Design

## Explanation

I have split the general design of my program into separate sections that I can program individually to assist with the development management of the project. To start with, I have split the main application into 3 parts that will combine to make the main part of the program.

Password randomiser - I will start by making a password randomiser that has a set of criteria for making the strongest password possible. I will achieve this by assigning a list of characters to each kind of character such as a list of capital letters and a list of symbols. From my research I found that a strong password is considered to have at least 1 capital letter, contain at least 1 symbol, contain at least 1 number and be a length of at least 12 characters long. I can use the random feature of python to put a random list of the types of characters together and make a check to make sure that all of the special characters are included. if it does not include a symbol, number or capital letter then the program will put the required character type on the end. It will the look through each of the lists and supply a random character to each of the character types to form the password.

Login system- To save the login details I will have the program request the user to either sign up or sign in to the program. In order to sign up, the user must enter a username and password which will be saved into a separate CSV file to the one used to store the hashed passwords. The password input by the user will be hashed and then saved. After the login details have been saved, the program will compare the details input by the user to the list of details stored in the login file and either log in, or display that the username or password is incorrect.

In order to store all of the data for the accounts, I will use a CSV file and store the data as a spreadsheet. I can then use python to call the CSV file in order to fetch data from it. I will use a CSV file because the data can be checked easily and it can be manually edited using Excel to assist with bug fixing. The data will be stored in a table with “website name”, “username/email” and “password” as the columns on the table. All of the passwords stored as hashed text and can only be accessed using the master password.

To encrypt the password, I will use a symmetric ASCII hash that changes the ASCII value of each character by the same amount each time. This process can then be reversible when the master password for the application is entered. This will run the hashing algorithm backwards to find the source text and then display it to the user ready to be copied.

For the password strength checker, I will give it the same requirements that the password randomiser uses to generate a strong password. These requirements being at least 12 characters long, containing at least 1 number, 1 capital, and 1 special character. If 2 or less features are included the application will output “weak”. If 3 features are included it will output “strong”. If all 4 features are included the output will be “very strong”. These ratings will then be added to the table where the account details are stored.

In terms of how the program will fit together I want to use a set of functions for each of the features in separate files. I decided on this because I think object-oriented programming will be far too time consuming especially since I won’t be using repeat objects very much. Using modular functions, it will be easier for me to track what is happening to variables which will help with bug fixing. It will also

## Key variables and data structures

|  |  |  |  |
| --- | --- | --- | --- |
| Variable | Data type | justification | Validation required |
| Master\_Username | String | The user needs to have a user name in order to have their account details stored to their own personal file. | Does the master username match the saved username? |
| Master\_Password | String | The user needs the master password in order to both log in to the application and then | Does the master password match the saved master password for the given username |
| Password\_Length | Integer | This will be the variable that determines the length of the users randomly generated password. | What is the value of the integer that the user input. |
| Characters | List | This will contain the types of characters available to the algorithm in order to create a (letter, capital\_letter, number, special\_character)random list of the types of characters | does another character need to be added to the |
| Character\_list | List | An empty list that the character types will be placed into randomly = to the length of the password | does another random character type need to be added to the list |
| number | List | The list that contains all of the numbers available to be generated in the password (0-9). | does the random character needed in the completed list = number |
| letter | List | The list that contains all of the letters in the alphabet (a-z) in lower case | Does the random character needed in the completed list = letter |
| Capital\_letter | List | The list that contains all of the letters in the alphabet (A-Z) in upper case | Does the random character needed in the completed list = capital\_letter |
| special\_character | List | The list that contains special characters that can be used in passwords e.g. !, $, or # | Does the random character needed in the completed list = special\_character |
| Completed\_list | List | The final list used in password generation that will hold the character accumulated through random generation that can be combined to form a string | Does the number of items in completed\_list = the number of items in character\_list |
| Score | Integer | The score assigned to a password when a strength check algorithm is run on an input. The score ranges from 0-5 with 0 being invalid and 5 being | Is the number greater than 0 and less than 6 |
| Encrypted\_password | string | The password after it has been encrypted using the ROT-13 encryption algorithm. | Does the encrypted password have the same length as the input |
| Decrypted\_password | string | The variable use after the encrypted password has been decrypted so it can be displayed in a text box | Does the decrypted password have the same length as the encrypted password. |
| Website\_name | string | The Variable that will link to the website name column in the saved CSV file | Was a valid input entered into the input box |
| Account\_name | String | The variable that will link to the account name column in the saved CSV file. | Was a valid input entered into the input box |
| New\_password | String | A variable that will be used to save a new master password to the master password file | Was the new password entered a valid input (i.e. not left blank) |
| Length | Number | The varible that stores the length of the password in the strength check algorithm so that it can check if the length is 12 characters or greater | Is length = to the number of characters in the password |
| Strength\_list | list | A list that stores each character type in the given password so that each type can be added to the score. | Is each item in the list = to one of the character types |
| Password | String | The variable that links to the location of the password column in the saved passwords CSV file. This column is where all of the encrypted passwords will go. | Was the input entered into the input box valid |

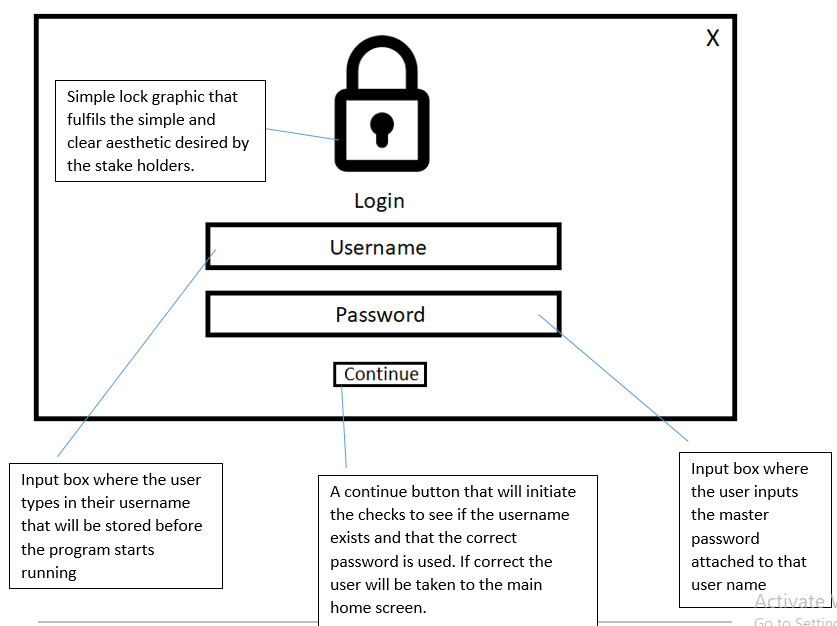
## Test Data for Development

|  |  |  |  |
| --- | --- | --- | --- |
| Test | Inputs | Expected result | Justification |
| Encryption | “Password” | I expect each letter to be converted into an ASCII value and displayed in a list. Then I hope to see that a second list is created with each ASCII number | The encryption is a key feature of security with the program and if it does not work properly, it will cause major problems with saving and copying passwords across |
| Decryption | The output of the password encryption test | I expect to see the list have 13 taken away from the ASCII characters and then for the output to be “Password” | The decryption needs to work so that the passwords can successfully be extracted from the program |
| Random password generation | “12” | I expect to see a password of random letters and characters that has a length of 12 characters | Having random generation of a desired length was part of my success criteria |
| Character diversity in the generated password | “4” | I expect to see a 4-character long string that has all 4 types of the different characters each time I run the program. I will repeat this 3 times to make sure it works properly. | This test is necessary because the generator needs to generate a password of at least a score of “strong” every time it is used |
| Make sure “invalid” works on password strength check | Leave the box blank | I expect to see the program tally up the points and total 0 followed by a score of “invalid”. | The password rankings are a key part of this program in terms of testing their strength. Making sure each score works is important for making sure they can be scored properly |
| Make sure “weak” works on password strength check | “Abcd” | I expect to see the points total be 2 followed by a score of “weak”. |  |
| Make sure “not bad” score works on the password strength check | “Abcd7” | I expect to see points total of 3 followed by a score of “not bad”. |  |
| Make sure the “strong” score works on the password strength check | “Abcd7%” | I expect to see points total of 4 followed by a score of “strong” |  |
| Make sure the “very strong” score works on the password strength check | “Abcd7%efghijk” | I expect to see points total of 5 followed by a score of “very strong” |  |
| Make sure the master password has been saved | Personal password | I expect to see a file be made called “master password” which contains the hashed version of the master password. (Note. I will not know if the password was hashed successfully until the interface is complete and I check that the full system works) | The master password locks out important features of the program. Making sure the hash works properly and it is saved in the right way means that I can then access the locked features. |
| Make sure the master account name has been saved | “AlfieB” | I expect to see a file made called account name with the name “AlfieB” | The same with the master password the master username is a barrier to access locked features like decryption making sure it saves in the right format and that I can access it again is vital for the program |
| Make sure the log in system works | AlfieB  Personal password | I expect to see a popup box open with an input box for both username and password. When I enter my two inputs, I expect to see the output “correct” | This test acts not only as a login check but simultaneously checks that the password hash works as it should. I wont be able to test the hash until I can log in |

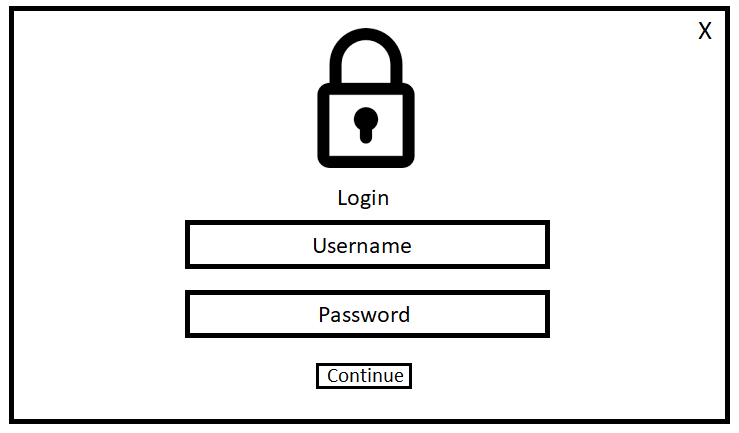
## Post development testing

|  |  |  |
| --- | --- | --- |
| Criteria | Measurement/ Evidence | Justification |
| Storing multiple passwords | The user should be able to store at least 3 passwords in a clear locked file. | This is a key feature of the program that I am determined to complete because it is a key feature of all of the other programs and is the whole point of the program |
| Strong password randomiser button | The user should be able to generate at least 5 12 character passwords that would get a “Strong” rating when using the built in password checker. | The password generation feature is important for the users who would otherwise struggle to create strong passwords on their own. |
| Encrypting and decrypting stored passwords | The user should be able to store 3 encrypted passwords in a locked file that do not resemble the original passwords. All 3 should then be successfully decrypted when the copy button is used. | This is a very importamt area to ensure that that the program is secure as well as this other parts of the program will rely on the passwords being encrypted and decrypted correctly. |
| Working copy button | The user presses the copy button which copies the encrypted password and decrypts it and save the decrypted text to the clipboard. 3 different passwords should be successfully copied and pasted to see if it matches the original password. | A useful user experience feature that will make copying passwords accross from one box to another faster and more efficient. |
| Multiple accounts management | The user should be able to create at least 2 accounts that have 2 different databases of stored passwords and account details. | This is important for if there are multiple users who want to use the app on the same device. Ensuring that the individual account files don’t mix is important for making this work |
| Working password strength checker button | The password strength check should give a valid result for at least 5 passwords (valid result: a result that matches to the checklist of what makes a strong password). | Making sure that the strength checker is consistent and accurate to the specifications will be important to give users the additional security that they are using strong passwords. |
| Log in boxes work | The user should be able to type in the given details correctly to log into their specific account | This is important to make surte that the encrypted password stay hidden from anyone who is not meant to access them |
| Clear and concise design that is easy to interpret for all users | The user should be able to access all features of the program without given help or advice | This includes features such as images that help the UI to be more clea and concise so that the user can navigate it easily |
| Passwords should be stored along with a website name and an account name | At least 3 full rows filled out complete with password, account name, and website | This will help the user to find the right account and subsequent encrypted password faster |

## Sketch of screen layout

Login screen (the initial screen displayed when the program is opened)

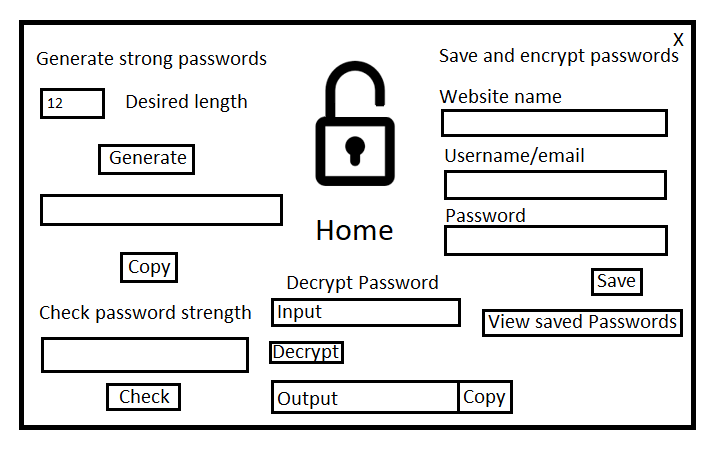
Simple lock graphic that fulfils the simple and clear aesthetic desired by the stake holders.



A continue button that will initiate the checks to see if the username exists and that the correct password is used. If correct the user will be taken to the main home screen.

Input box where the user inputs the master password attached to that user name

Input box where the user types in their username that will be stored before the program starts running



12

13

11

10

9

8

7

6

5

4

3

2

1

1. Simple graphic of an unlocked lock symbolising that the user is free to use the program this also helps with the simplistic aesthetic of the program.
2. An input box for the user to input the desired length of the password.
3. A button that causes a random string generation algorithm to generate a password that can be copied to the clipboard and saved to the CSV file.
4. Display box where the password will be displayed.
5. An input box where the user inputs their desired password so it can be checked by the strength test algorithm.
6. A button that runs the strength test algorithm and displays the output as a popup message.
7. Input box that will be the input for the decryption part of the program
8. A button that will reverse the encryption algorithm to decrypt the text input and then display the output in the output box (this will require the master password to use)
9. Output box where the decrypted password is displayed.
10. A button that copies the text in the output box to the clipboard
11. The input boxes for the storage part of the program labelled so that it is clear to the user where each personal detail should go. Each box corresponds to a column in a CSV file.
12. This will take the text from each of the input boxes and put them into a column in the CSV file. With the “Password” box the text will be hashed before being placed into the CSV file for added security
13. This will request the user to enter the master password. If it is entered correctly, the user will be taken to the CSV file where the account details and hashed passwords are stored.

A screenshot of a spreadsheet

Description automatically generated

Encrypted passwords

Usernames

Website names

This is a plan for the layout of the saved account names and passwords. Column A is where website names will be stored as seen above. This is where the website name input box on the main interface will link to. Column B is where all the usernames are stored, and this is linked to the username input box on the main interface. Column C is where I ran a test algorithm for encrypting the passwords and stored them for each attached username. The reason why I want to store these as a CSV file and use Excel is because it has a built-in keyword search function which will enable the user to quickly find saved usernames and passwords.

## Key Algorithms and Framework

Algorithm for the encryption and decryption

I will for the cipher I will use a slightly altered version of the ROT13 cipher. This essentially works by moving each letter 13 places along in the alphabet and then redisplay the message with the new letters. I want this cypher to incorporate the symbols as well. There are 2 ways I can achieve this the first way is using a large dictionary that assigns an integer value to each character. However, this method would be very time consuming and could cause problems later down the line with storing the values. The other way I could achieve the ROT13 cypher including numbers and symbols is to use ASCII. Python has an inbuilt function that can convert a character to its ASCII value. Using this method, I can separate the string into a list and convert each character to ASCII one by one and then add 13 to the ASCII value then convert it back into characters. This method uses the pre-existing ASCII library which will provide a much more efficient solution to this problem. The issue with using ASCII is that the value is limited to 255. This is a problem because if the final ASCII value exceeds 255 then the algorithm will not be able to convert the ASCII into a character. This should not provide an issue because the range of characters available on the English keyboard does not exceed the 241st ASCII value.

Define encrypt:

num\_list = empty

encrypt\_list = empty

for character in password:

y = change character 🡪 ASCII + 13

add y to num\_list

print(num\_list)

for integer in num\_list:

z = change ASCII 🡪 Character

add z to encrypted\_list

print(encrypt\_list)

encrypted\_password = join(encrypt\_list)

return encrypted\_password

define decrypt:

num\_list2 = empty

decrypt\_list = empty

for c in encrypted\_password:

t = change character 🡪 ASCII - 13

add t to num\_list2

print(num\_list2)

for integer in num\_list2:

u = change ASCII 🡪 Character

add u to decrypt\_list

print(decrypt\_list)

decrypted\_password = join(decrypt\_list)

return decrypted\_password

Algorithm for Password generation.

For the password generation algorithm, I need to find a way to generate a string of completely random characters. I think the best way to achieve this is to generate a list of random characters and then join them to form a string that can be displayed to the user. To generate this final list, I need to make a define the difference between the main characteristics that make up a strong password. This being lowercase letters, uppercase letters, numbers, and special characters (e.g. £, <, %, #, etc.). To do this, I will create separate lists and assign them names such as “upper\_letter, lower\_letter, number, and special character” with these for lists I can generate a random combination of each of these types of characters equal to the desired length of the password. I can then have the algorithm run through each item in the list and assign a random character to each of the character types. I can then join the list together and form it into a string that will be displayed to the user.

Lower\_letters= [(letters a-z)]

Upper\_letters= [(letters capps A-Z)]

Numbers= [(numbers 0-9)]

Symbols=[(all symbols available on the keyboard)]

Characters = [ Lower\_letters, Upper\_letters, Numbers, Symbols]

Character\_list= []

Pswd\_list = []

Pswd\_length = (integer obtained from user)

For x in range(pswd\_length):

X = Random.choice(Characters)

Character\_list.append(x)

For x in character\_list:

If x == ‘lower\_letters’:

A = random.choice(lower\_letters)

Pswd\_list.append(A)

If x == ‘Upper\_letters’:

A= random.choice(Upper\_letters)

Pswd\_list.append(A)

If x == ‘Numbers’:

A= random.choice(Numbers)

Pswd\_list.append(A)

If x == ‘Symbols’:

A= random.choice(Symbols)

Pswd\_list.append(A)

Full\_password = pswd\_list.join

Print(Full\_password)

Password strength check algorithm.

For testing strength, I will need to score passwords based on certain criteria that will determine how safe a password is. The scoring will be based on length, character diversity, upper case letters, lower case letters and numbers. The score will go from 0-5 with one point being awarded for each fulfilled criteria.

Because I will have already made a list of all the characters for the password generation algorithm I can use the same lists for scoring the passwords.

Strength\_list=[]

score = 0

length = 0

pswd\_list = list(password)

for x in pswd\_list:

if x in lower\_letters:

strength\_list.append("lower\_letters")

elif x in upper\_letters:

strength\_list.append("upper\_letters")

elif x in number:

strength\_list.append("numbers")

elif x in symbol:

strength\_list.append("symbols")

if "lower\_letter" in strength\_list:

score = score + 1

if 'upper\_letter' in strength\_list:

score = score + 1

if 'number' in strength\_list:

score = score + 1

if 'symbol' in strength\_list:

score = score + 1

for x in strength\_list:

length = length+1

if length > 11:

score = score + 1

strength\_list.clear()

print(score)

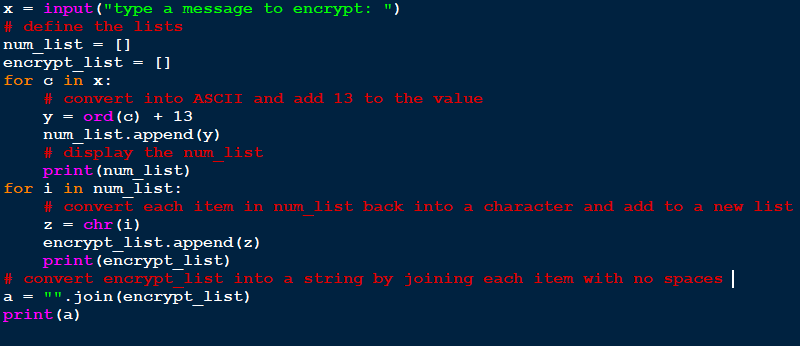
to bring these key algorithms into the main interface I will make them all as functions in separate files that I can then import into the main file. This will help to keep the main file clean and less messy as well as helping me to find the algorithms more efficiently.

# Developing the Coded Solution

## Development checklist

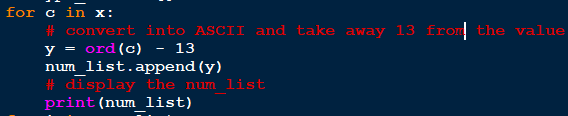
1. Working cypher (done)
2. Working password generator (done)
3. Use the variables from the password generator to create a password strength checker. (done)
4. System that can store a master password and username. (done)
5. Method to hash the master password and username. (done)
6. Working login and new user system (done)
7. Main file that can be as the base to import the separate functions. (done)
8. Base GUI incorporated into the encryption and decryption algorithm. (done)
9. Base GUI incorporated into the password generation algorithm. (done)
10. Base GUI incorporated into the strength test algorithm. (done)
11. Test methods of account detail storage (done)
12. Link storage to the base python file (done)
13. Method to add the inputs from the python file to the external storage file. (done)
14. GUI for the account details storage (done)
15. Link the master password to each of the features that enable the user to see personal info. (done)
16. Test log in and new account generation and ensure the master password and username are stored securely.
17. Test password generation and ensure that it can be copied with the press of a button.
18. Test that the password strength test system works.
19. Test that the account storage works and that the external file can be opened upon the input of the master password for that user.
20. Test the encryption and decryption and make sure the passwords work and can be copied with the press of a button.

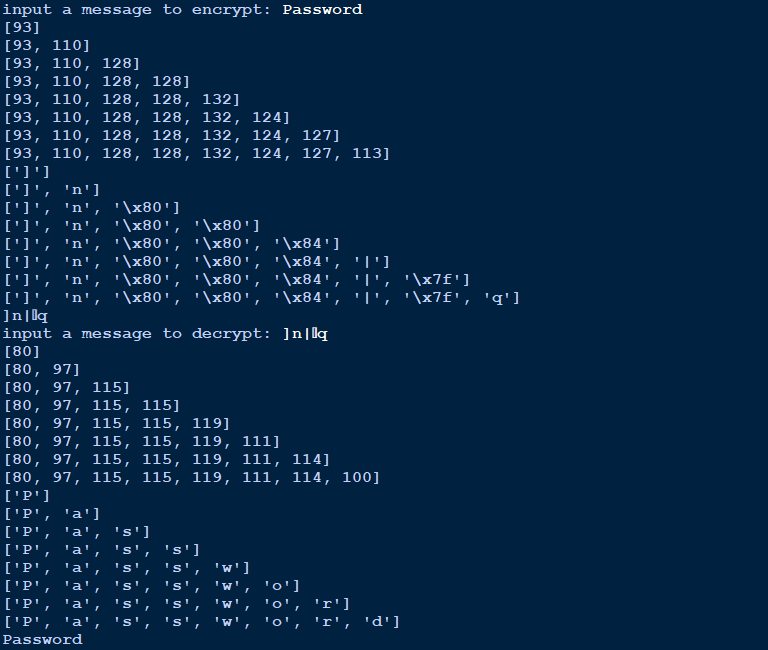
## Milestone 1: Cypher Algorithm

This is the first small algorithm I used to test that the logic of the cypher works. Similar to my plan I’ve used a series of lists to track where each character is in the process of being encrypted. Printing each result also shows me where the program is breaking if it doesn’t work in a certain area.

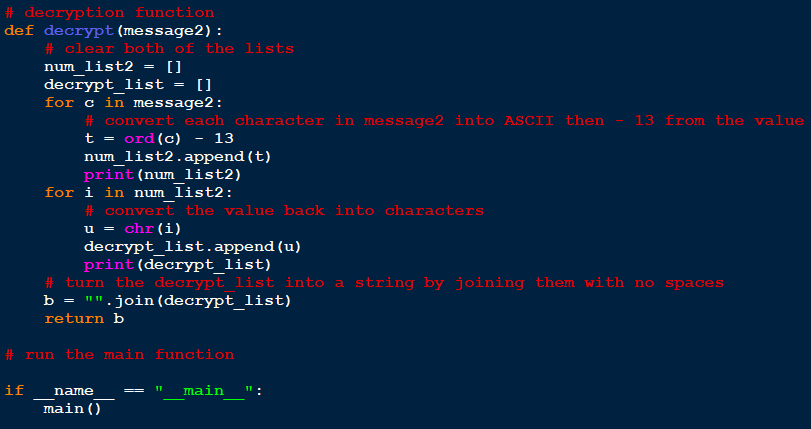


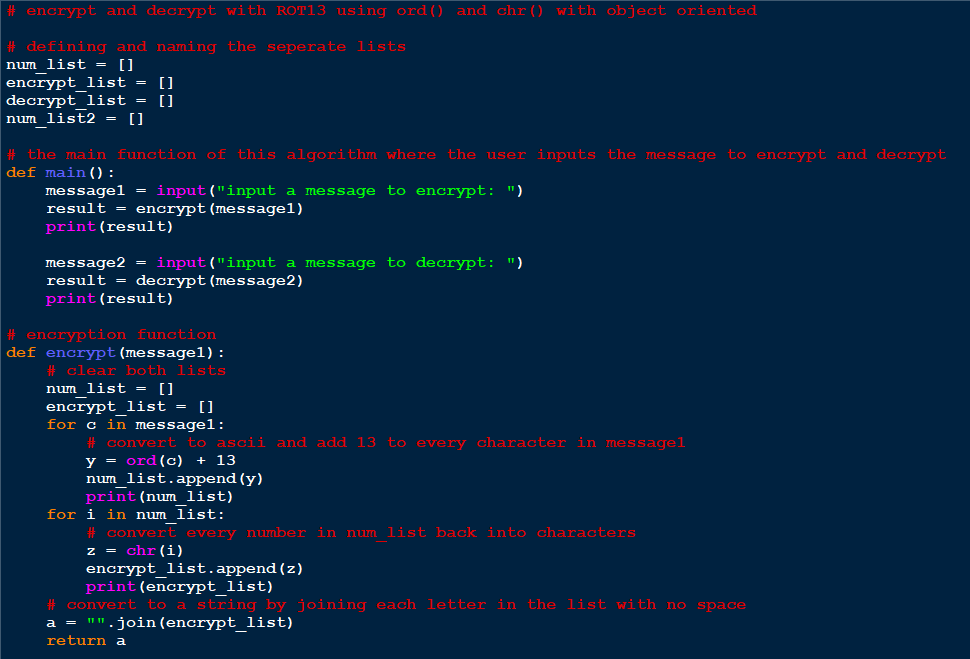
I ran this algorithm and input the word “Password” to encrypt. The output of this algorithm was “]n„|q”. This output was unexpected, so I found an ASCII table to see if the algorithm was correct. The first letter in the message was ‘P’ which holds the ASCII value of 80 if we look at the first item in the first list, we can see that the output is 93 which is 80 + 13. Which in theory means it output the correct value.

 I can test this by changing the addition constant from +13 to -13 to reverse and decrypt the output message.



I copied the original output and pasted it back into the input box the output as shown is “Password” which is my original input. This proves that the logic for the algorithm works so I can add classes and functions to it so it can be imported more easily.

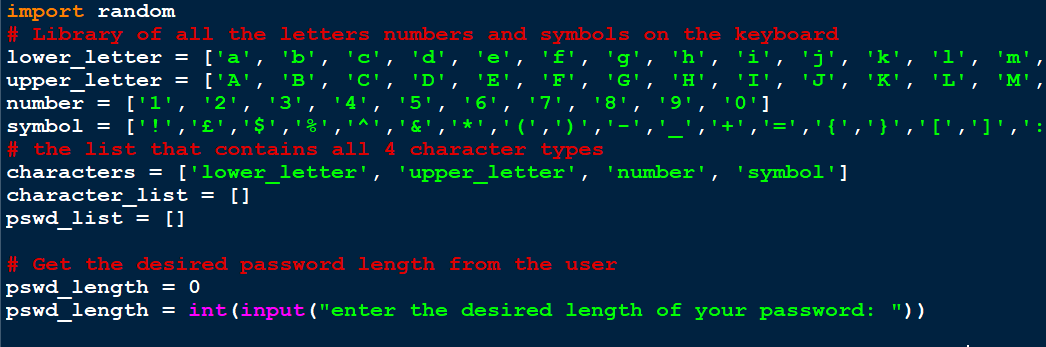


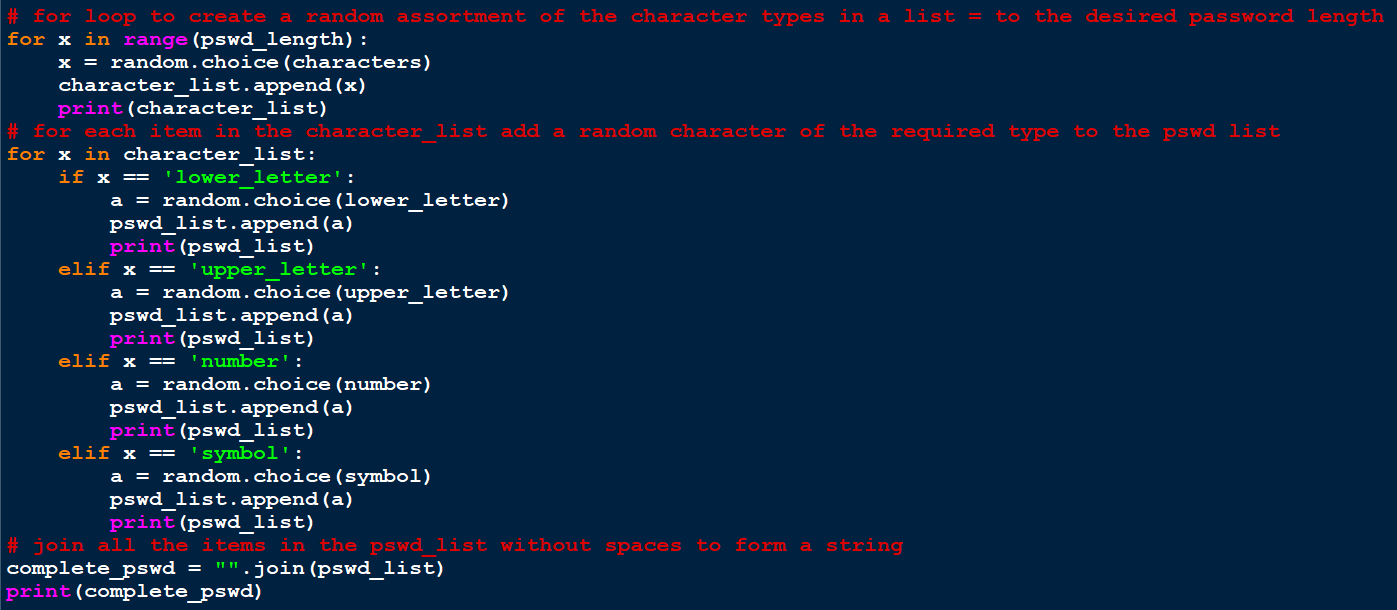


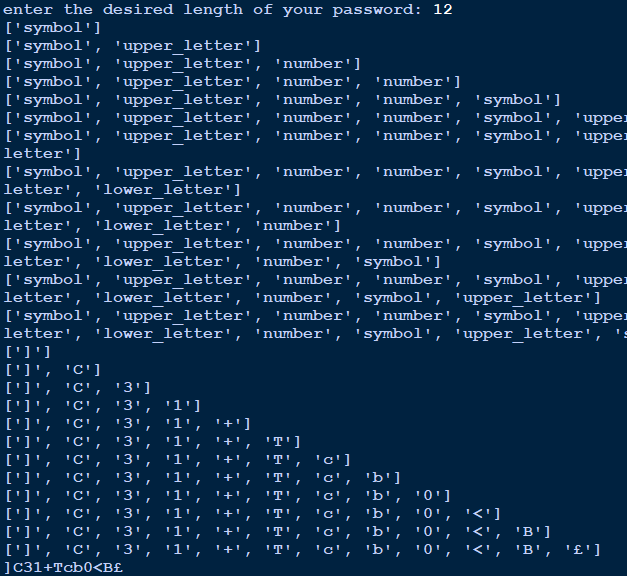
Using the same base logic as the test algorithm I added the two functions “encrypt” and “decrypt” to the algorithm. I had to make a few minor changes to the original algorithm such as having the functions call on the variables “message1” and “message2” and using return instead of print for the final output but it functions the same way as the test algorithm.

Now that I know the algorithm works I can remove the lines that print the lists after each step to make the program more efficient. This will save a lot more time since it is not trying to display every value it is working out. This will make the program much faster when dealing with longer strings.

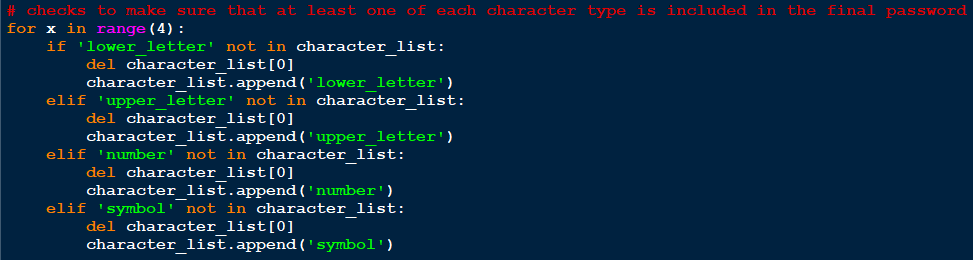
## Milestone 2: Random Password Generation

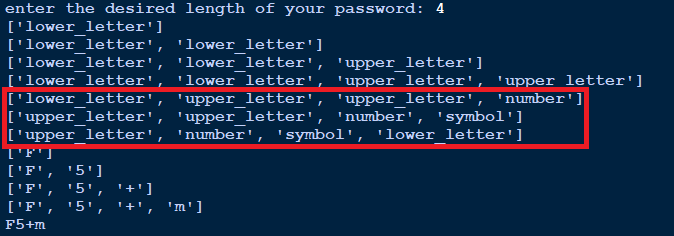


I started this algorithm by creating a list of all of the characters my program will accept and getting the pswd\_length variable which is a key variable for the program.

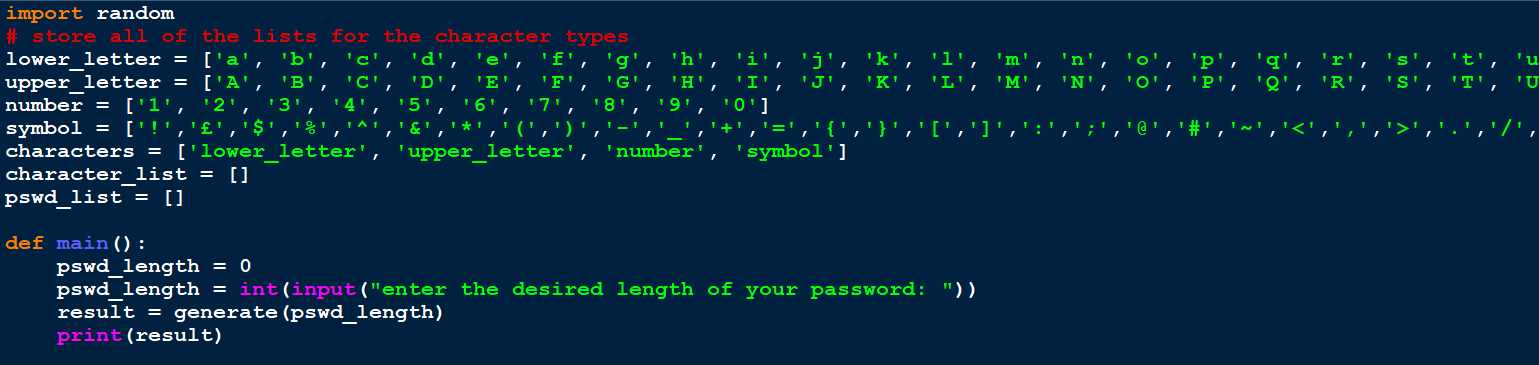
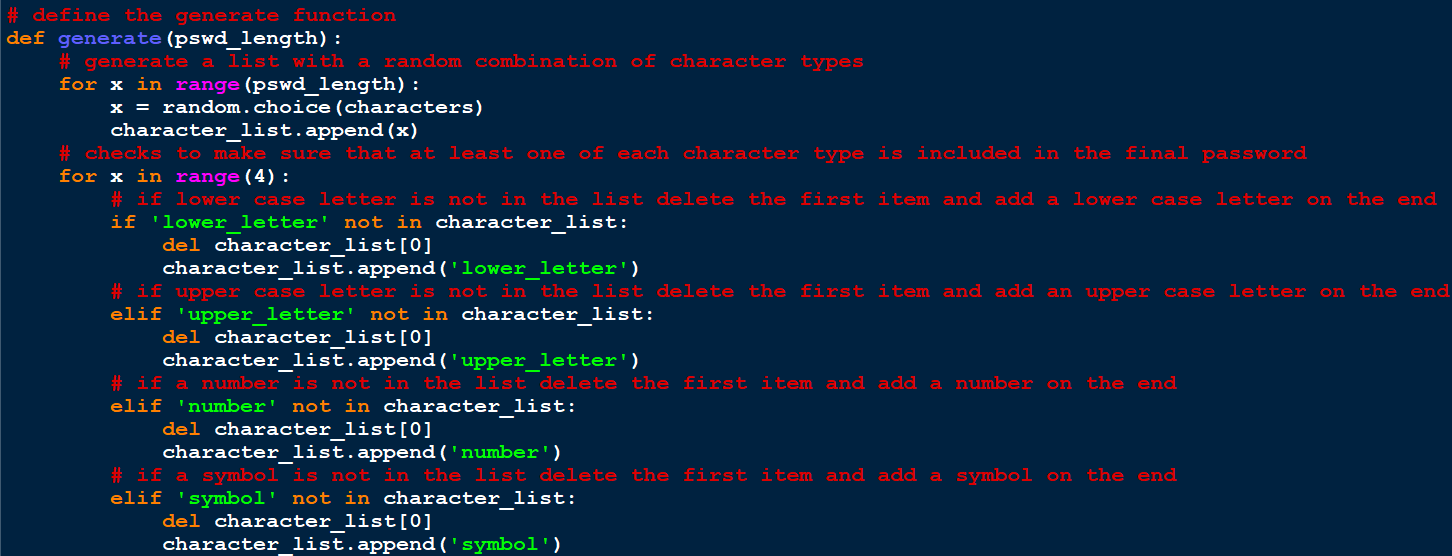
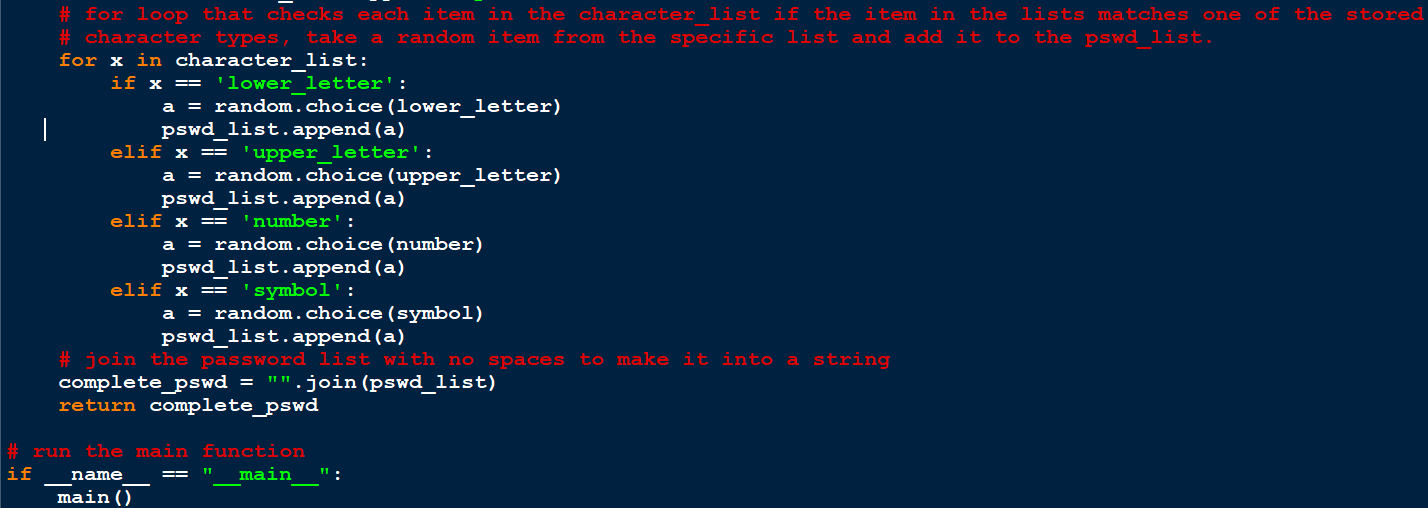
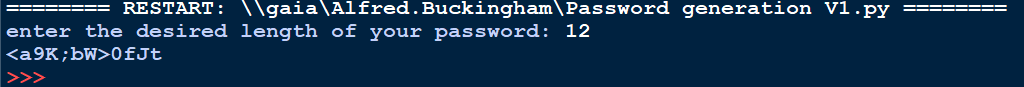
I then started the random generation with a for loop that will generate a random list of all the character types equal to the pswd\_length variable. The second for loop then assigns the characters to the final list where the algorithm will then join the items to form a string.

I ran the first test of the algorithm and it seemed to work well. This shows how the algorithm is generating the 2 separate lists and then using the lists to create a random string. An issue I have with this algorithm is that it does not guarantee every type of character will be in the generated password. This is a problem because one of the requirements of the password generator is that it will always output a “strong” password when the password checker is used on them. This means that the passwords have to contain at least one of every character. To fix this I will add another for loop to the algorithm which will run through the first list to make sure that at least one of each character type is included in the list. If it sees one missing it will delete the first item and slot the missing one on the end. It will then repeat this process until at least one of every type of character is included.

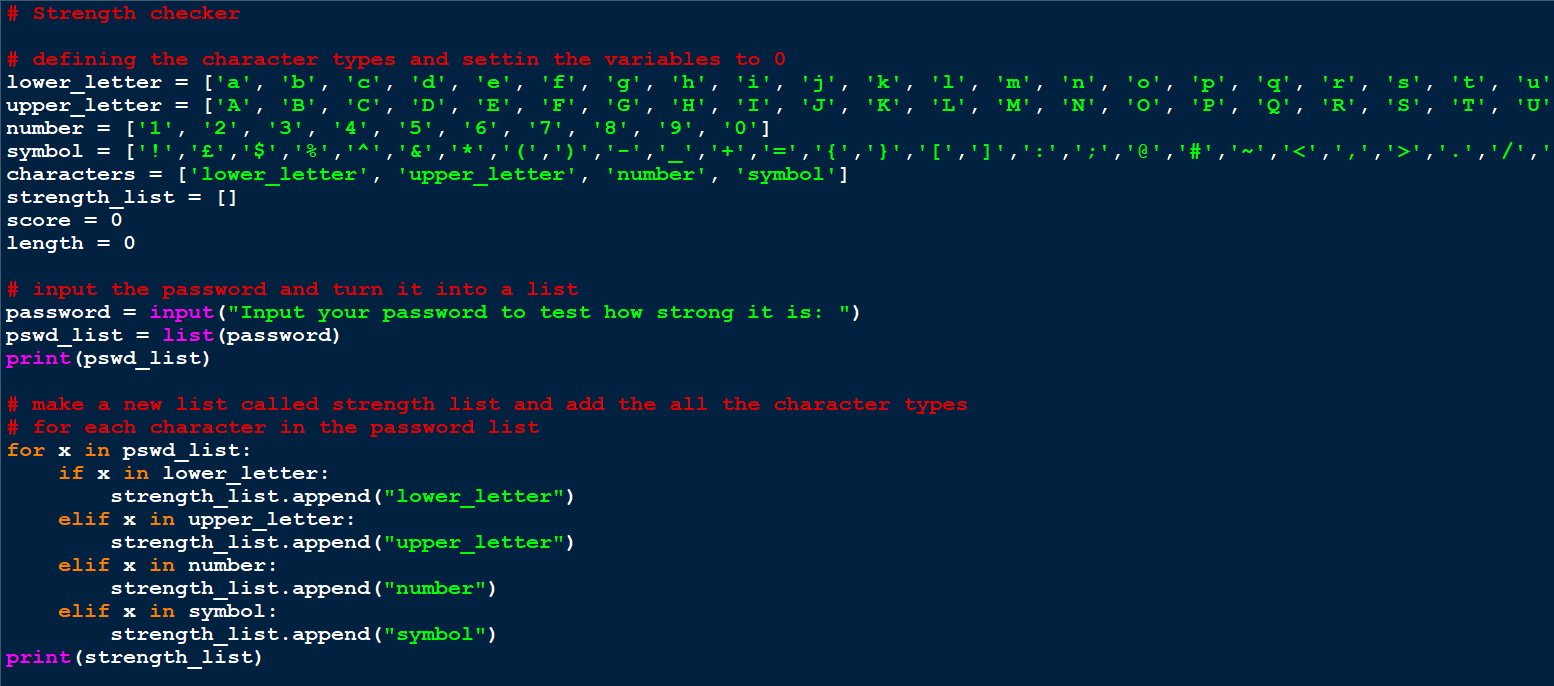
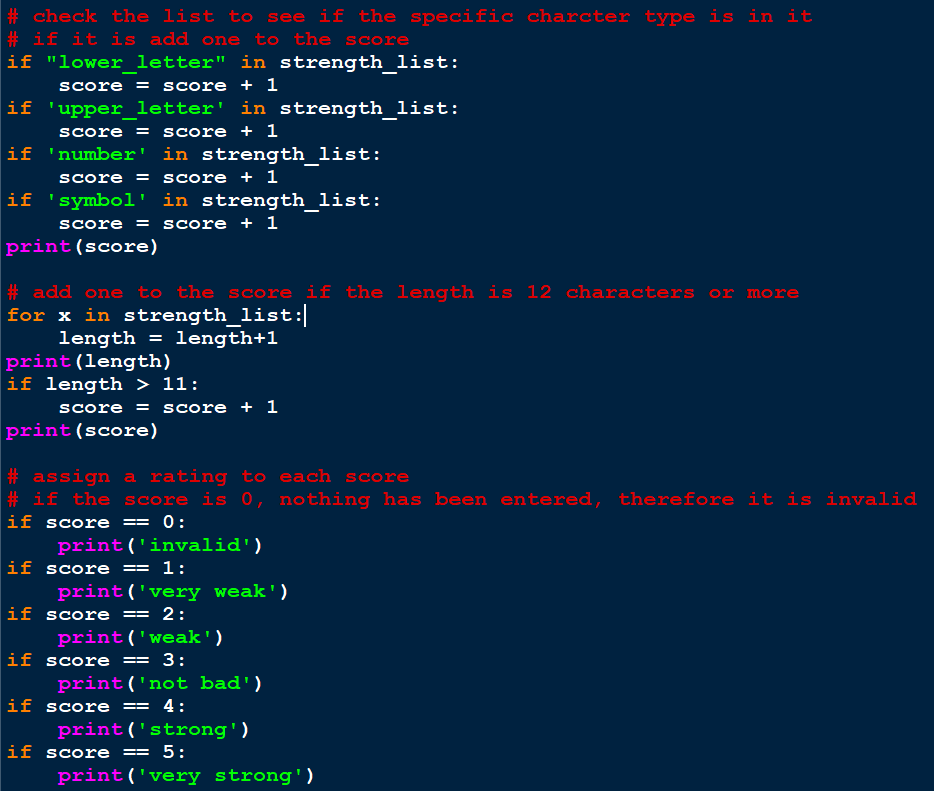
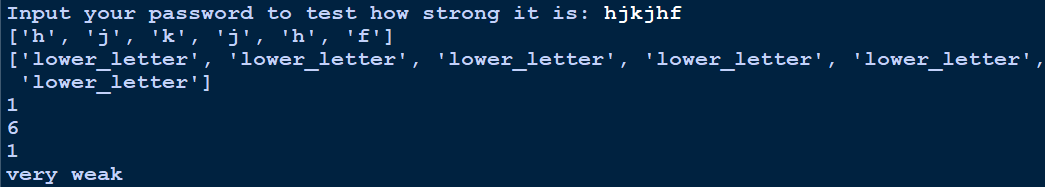
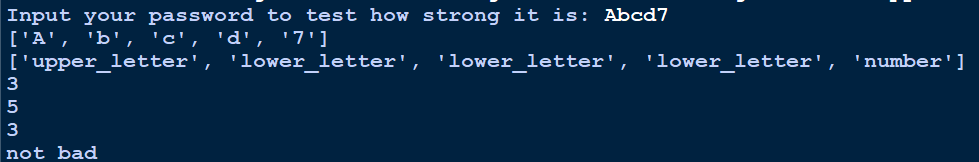
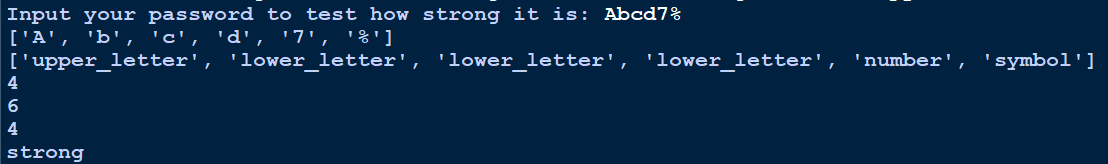
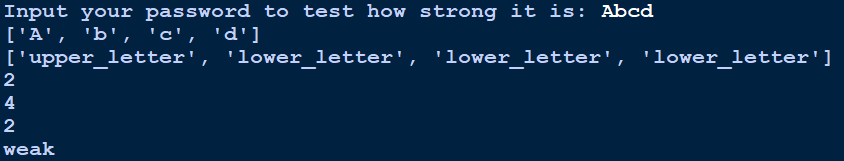
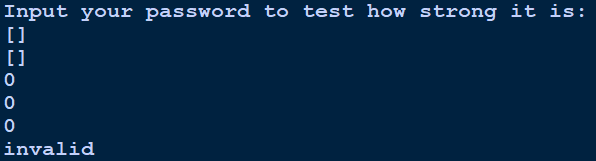
This is the first draft for the part of the algorithm that checks the initial list contains at least one of each type of character. To achieve this, it checks the algorithm 5 times. Each time it makes a check it is looking for one of the 4 types of characters that is not in the first list. The reason why it is checking 5 times is because 4 is the maximum amount of changes the algorithm will make to the list.

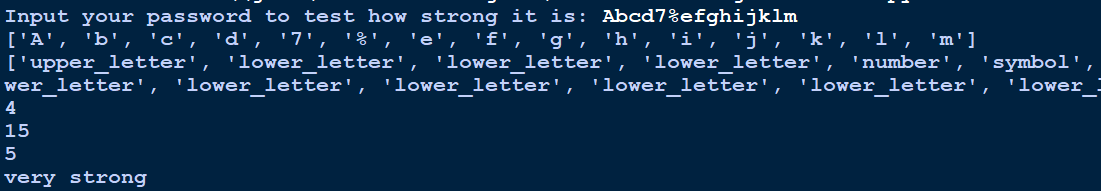
To test the improvements to the algorithm I included the printing of the list after each successful change. I input 4 for the password length since the initial random generation would likely have a repeat with only 4 items and as seen above 3 changes were made to the initial list and the final output contains all 4 types of characters. This proves that the password generation algorithm will always output a strong password.

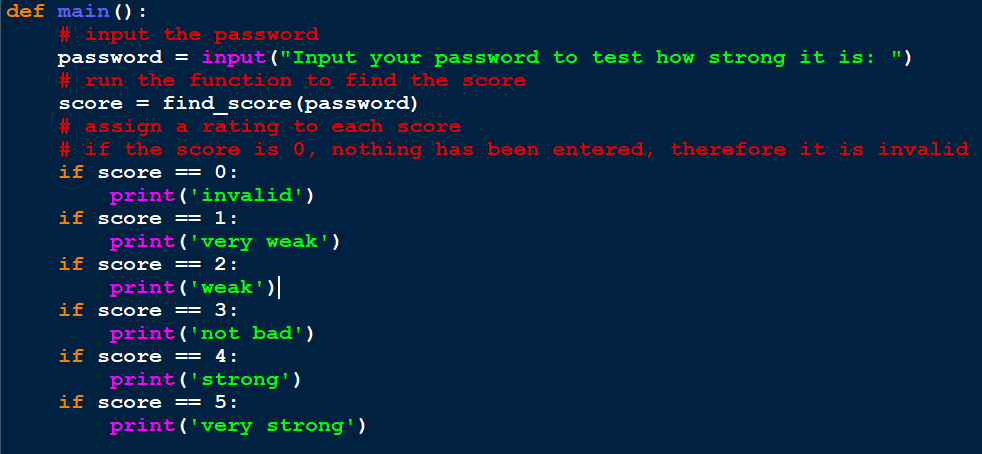
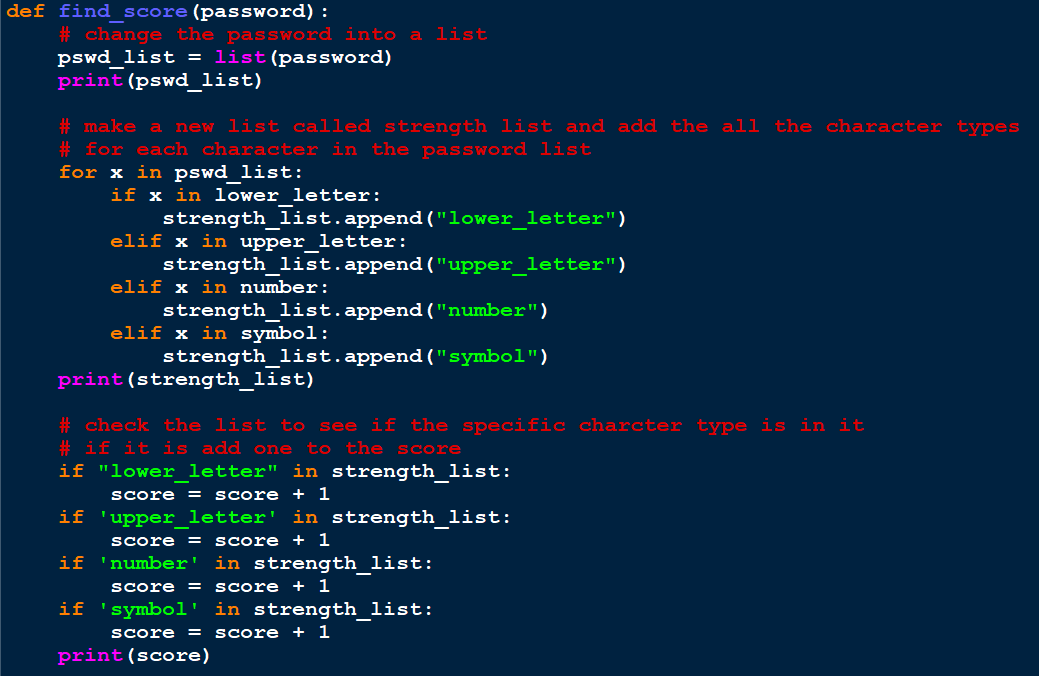
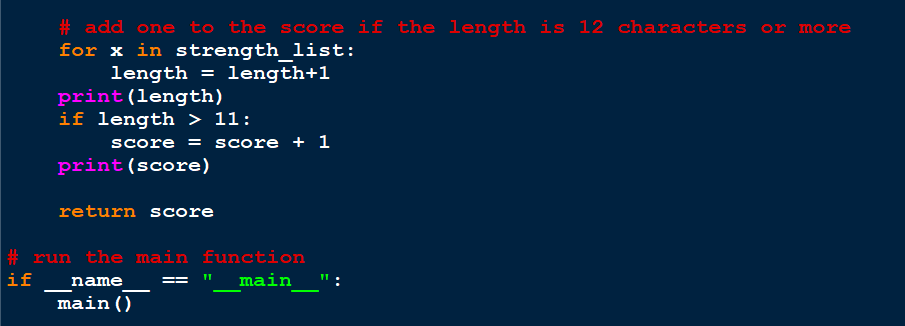
The next steps are to neaten up the algorithm and make functions for it so that it can be imported and run more efficiently.

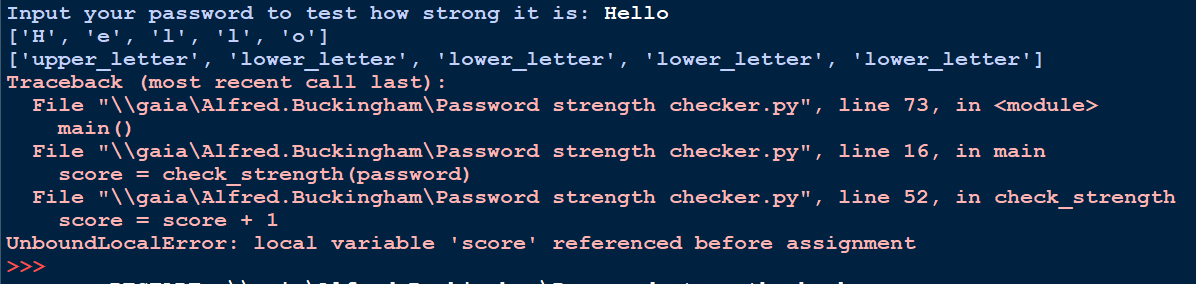
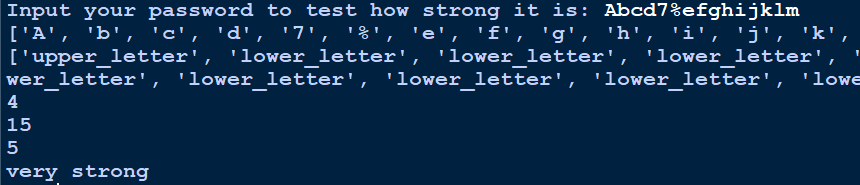
This is the complete algorithm for password generation which worked after doing a few tests I removed all of the print commands which made the generation much faster so it is now ready to be imported.

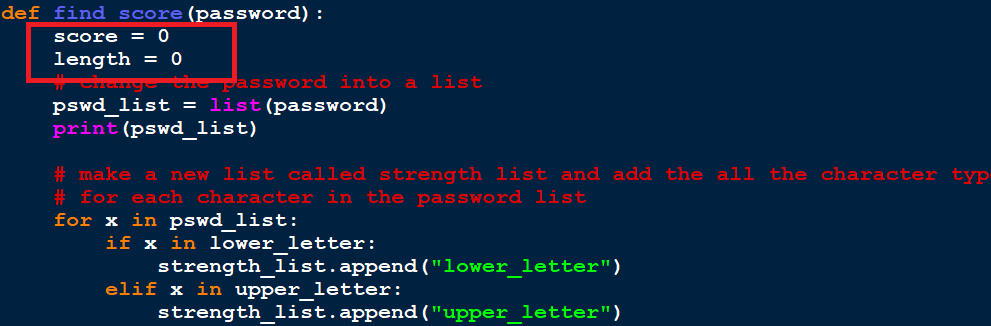
## Milestone 3: Password Strength Checker

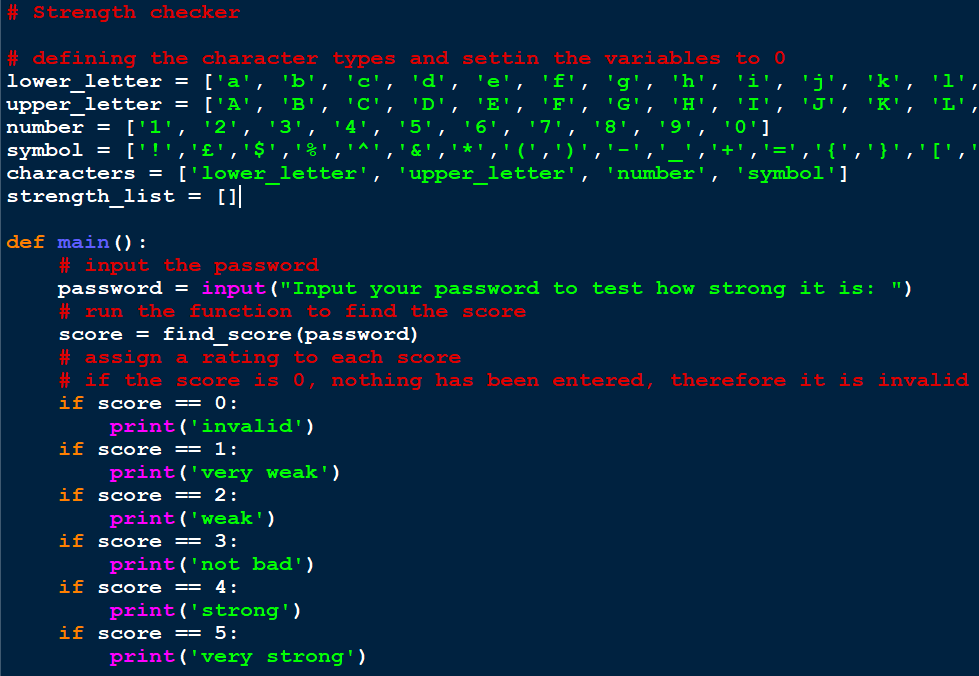
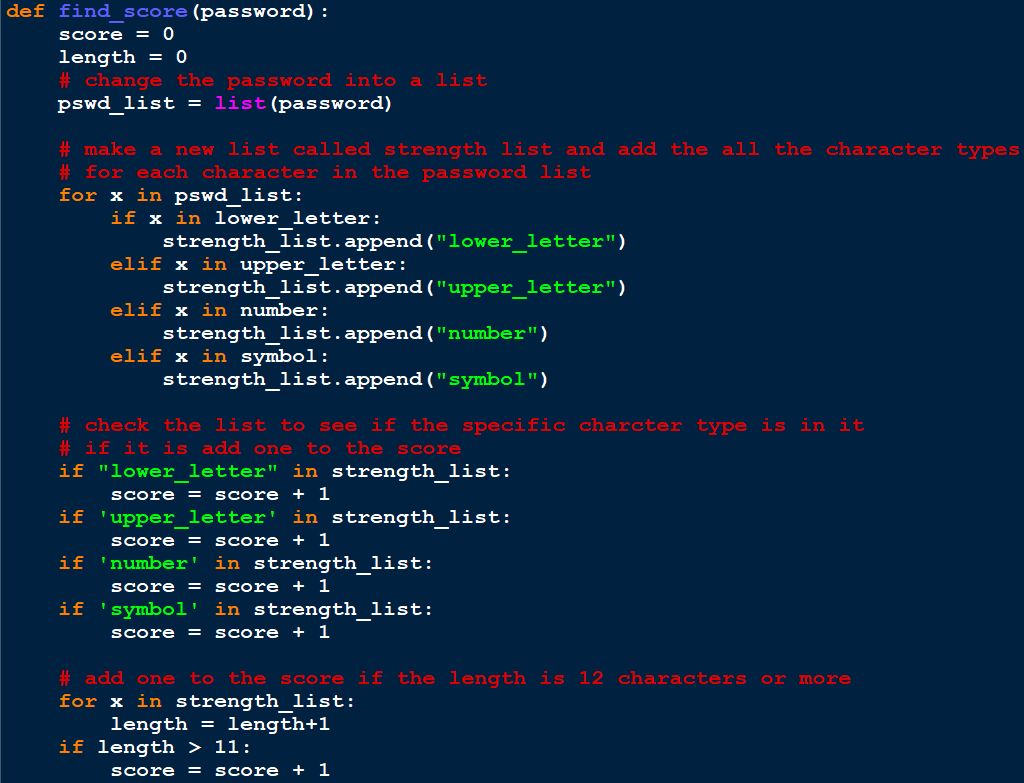
This is the first draft of the password strength checking system. I have used the same variables and lists as the password generator for ease of access when it comes to combing the separate files. The reason why the password generator guarantees a ‘strong’ password and not a very strong password is because the length is customisable. The criteria for the very strong password is to have at least one lower case letter, at least one uppercase letter, at least one number, at least one symbol and be at least 12 characters long. This algorithm assigns a score of one for each of the categories and then adds them up to find a final score. The final score is then assigned a rating from ‘invalid’ – ‘very strong’ (0-5).

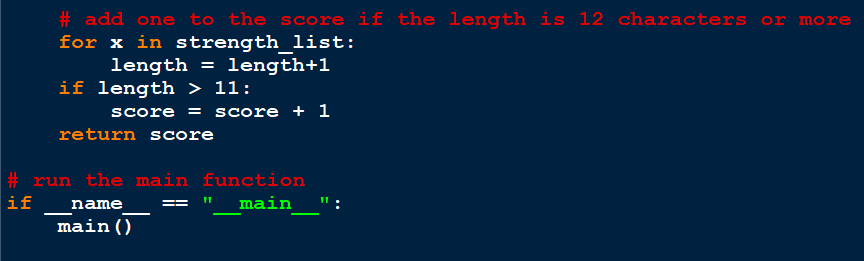
As seen above the password strength checker works according to the pre-set criteria with the 5 tests I did. I will take this test data and assume that it will work for all the passwords looking at how the lists are formed and how the score is accumulated. So will edit the program to change it into a function so it can be imported into the GUI file.

I changed the part of the algorithm which gathers the string into a list and obtains the rating into a function that finds the score of the string I can then return the score and use it to find the rating in the main function.

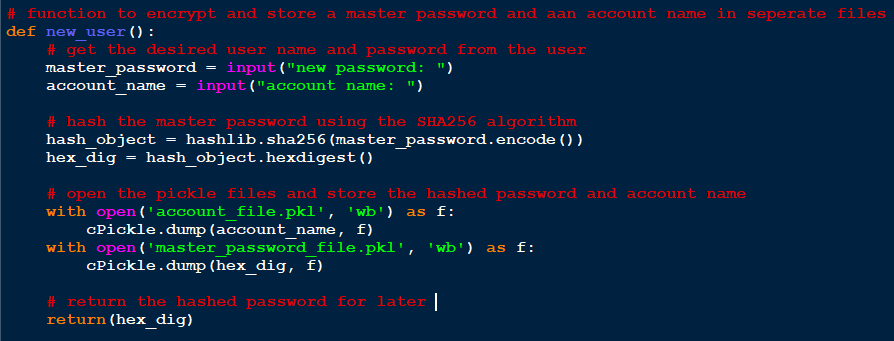
I ran a test just to check that the logic still worked, and I received this error. It says that the score variable is now local and has been used before the function. If it stopped at the score variable this means that the score and length variables will need to be set to 0 within the find\_score function to make them local variables.

I ran a final test using the same password I used in my test earlier and it came out with the same output using the updated algorithm with functions I can now delete all of the print commands since I know that the algorithm is working. This will make it more efficient especially when dealing with longer strings.



Final password strength checking algorithm ready to import.

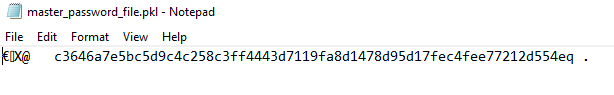
## Milestone 4: account name and master password storage



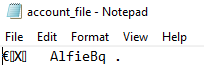
I made this small algorithm which runs in the python shell it asked for me to input my desired details and stores both in .pkl files which can then be opened using notepad. The password is hashed using the sha256 algorithm which ensures that the password I hashed into a string that is always 256 bits long. I was unable to create a full log in system due to time constraints but using this I can lock the program behind the password for a single user.



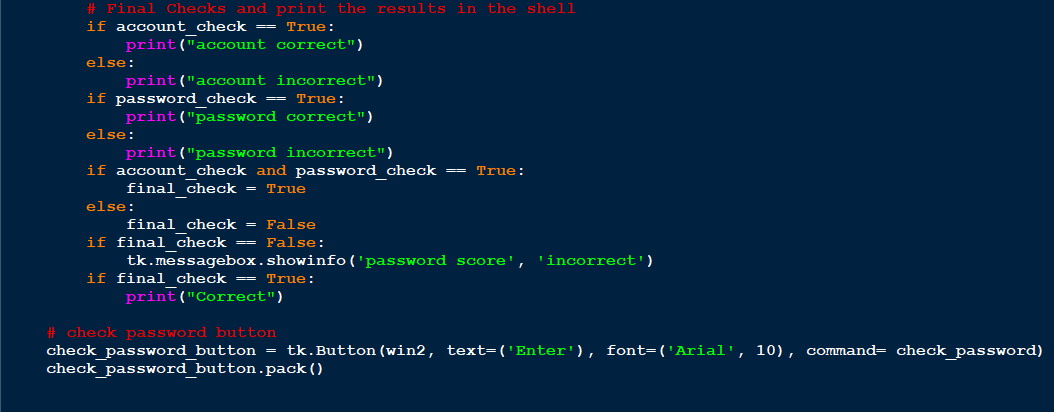
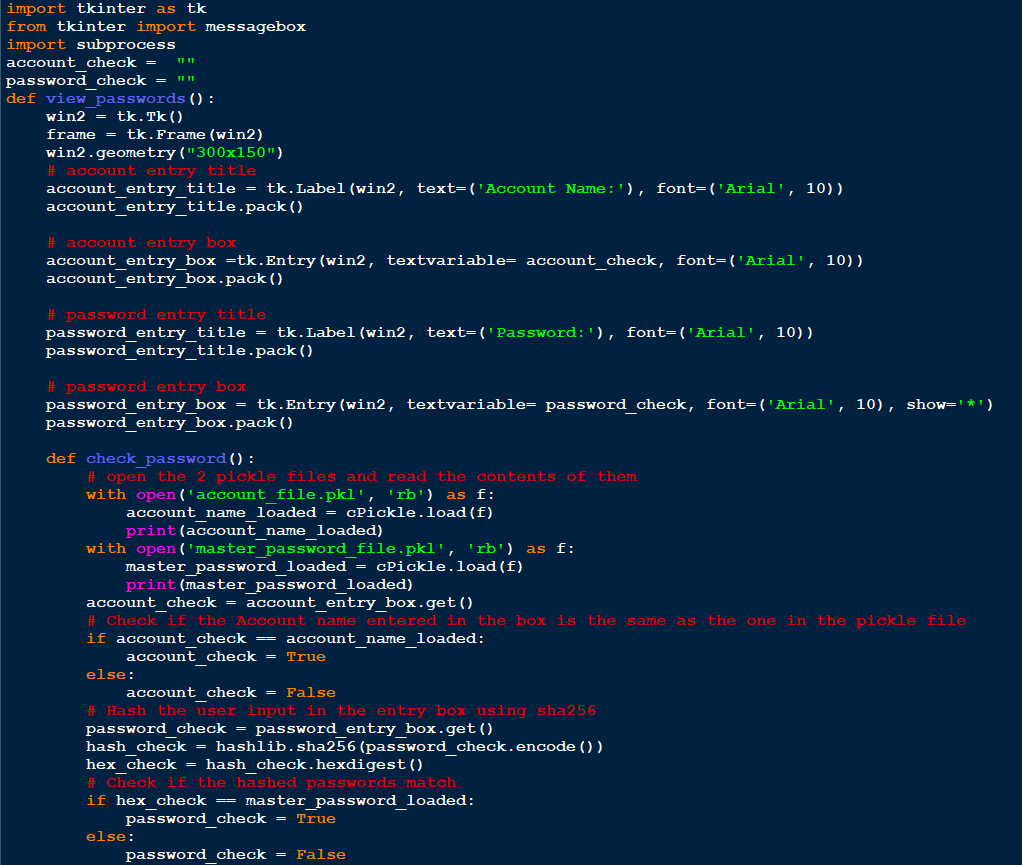


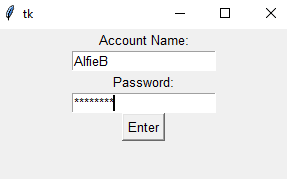


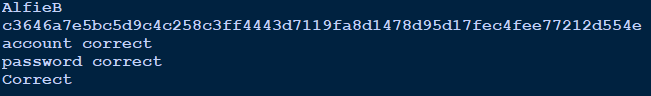
This is my personal password that has been hashed.



As seen above the program successfully created the 2 files and stores both the username and seemingly has hashed my personal password.

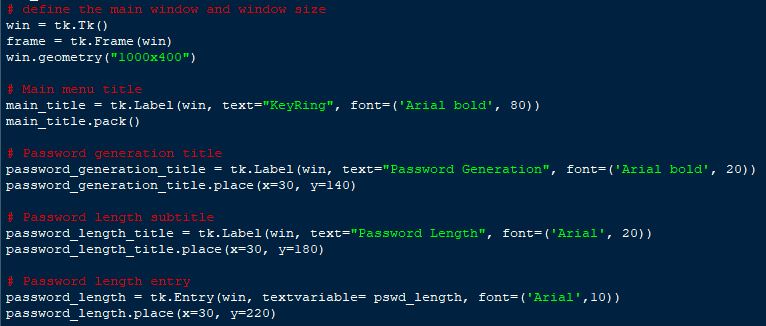
I won’t be able to tell if the hashing works successfully, so I created another function using Tkinter and entry boxes that hashes the input using the same sha256 algorithm and then compares the results.



This is the Tkinter window that appears when I run the function, so I entered my account name and my personal password into the appropriate boxes and clicked enter.

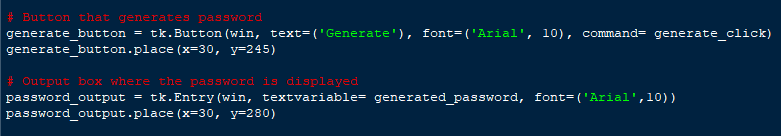
This was the result after clicking enter. The algorithm printed out the account name and the hashed password stored in the pickle files and then ran through the checks which all come through correct. This is a good result but just to make sure I re-entered the password incorrect on purpose just to make sure that the hashing algorithm worked properly.

## Milestone 5: Interface



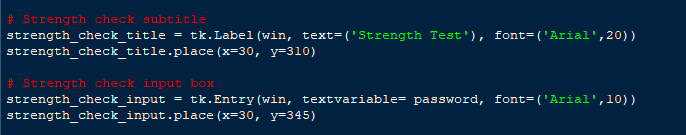
To start with I chose to use the tkinter library for the interface just like for the master password and account name. I started with the password generation side on the left side of the program and tested the placement and font sizes of the text and boxes. With each of the boxes for the generation now in the main file as seen above, I needed to add a generate button.

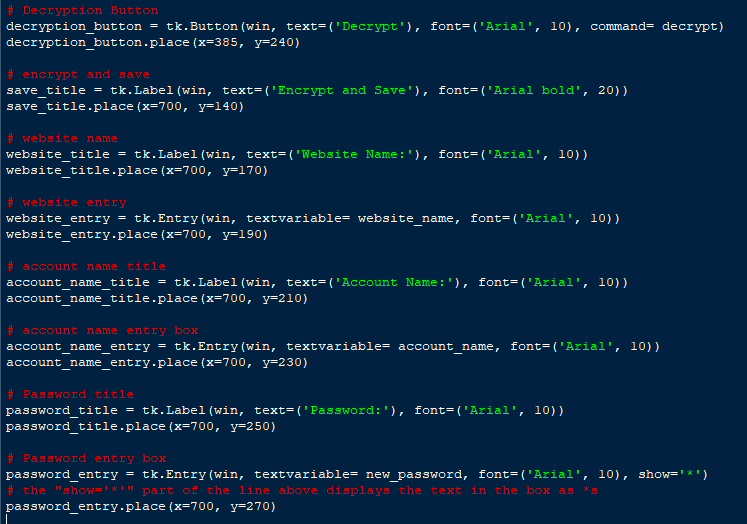
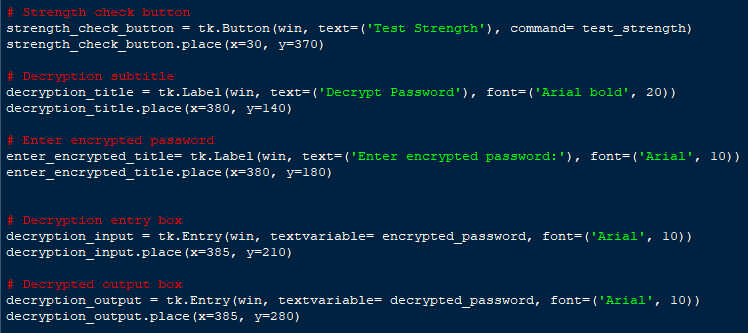
To get help getting started using tkinter to make an interface I used some of the articles from geeksforgeeks at [Python Tkinter Tutorial - GeeksforGeeks](https://www.geeksforgeeks.org/python-tkinter-tutorial/?ref=lbp).



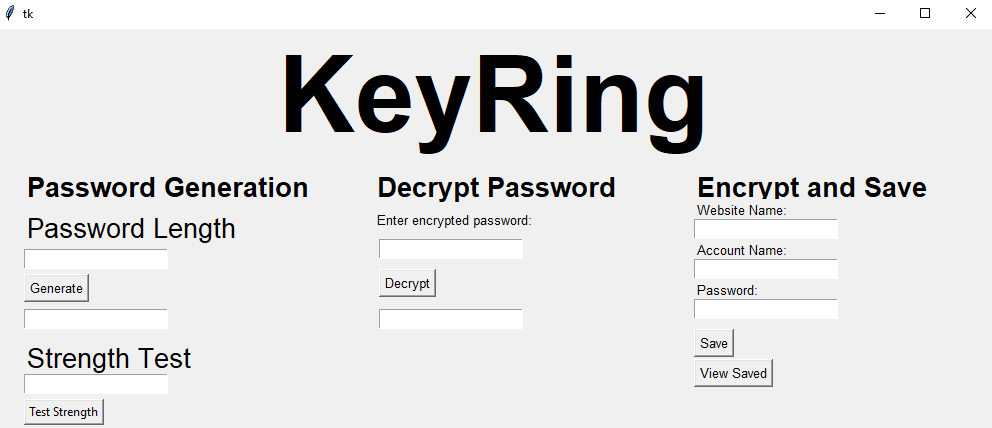
(figure 1)

This bit of code places the button and the output box. Because the button needs to do something it requires a command which I called generate click. This command requires me to make a function and this function is where I will import the password generation algorithm I made previously. The screenshot figure 1 is what the program looks like with that code. The boxes are not yet linked, and the button does not yet do anything, but I can run it and compare it to my plan to use as a visual guide when placing widgets with tkinter. As soon as the first widgets were placed I was able to follow the framework to complete the rest.





(figure 2)



I repeated the process of testing the x coordinates to make columns for each of the widgets using trial and error. Making sure I labelled each part, so I did not loose track when it comes to importing the different algorithms. The result of the trial and error with the coordinates look like figure 2. Unfortunately. I was not able to include the graphic I had made in the design stage due to requiring a library that was not available to me at the time of programming, but I did manage to keep the general layout fairly close to the design. Much like in figure 1 none of the boxes are linked and none of the buttons work yet but I am happy with the layout of the program which shouldn’t make it too difficult to import each algorithm.

## Milestone 6: importing the algorithms.

A screenshot of a computer

Description automatically generated

9.

8.

7.

6.

5.

4.

3.

2.

1.

I first started by compiling all of the algorithm and storage files into one main file. This way I can link everything into the main interface file which acts as the main application.

1. This is the file that stores the master account name for example AlfieB.
2. This is the CSV file I want to use to save and encrypt all the passwords, website names and account names.
3. The encryption and decryption algorithm set as functions ready to import to the main file.
4. The main menu interfaces. This will be the main file that all the other functions will be imported to.
5. The file that stores the master password which has been hashed.
6. The function that hashes the master password and stores it as well as the master account name.
7. The password generation algorithm
8. The algorithm I want to use to save all of the inputs of the “Encrypt and Save” area of the main menu and store in the columns of the CSV file.
9. The password strength checking algorithm.

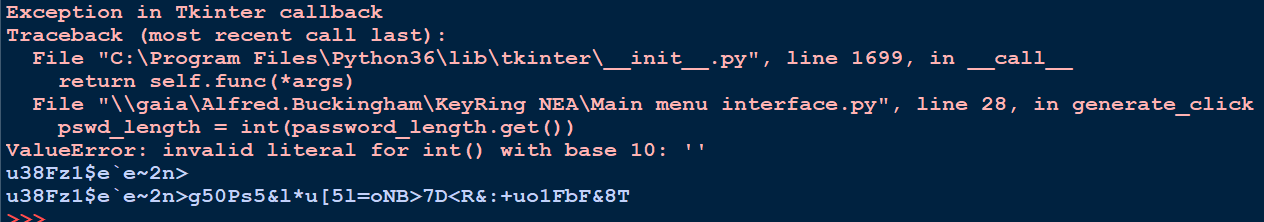
A screen shot of a computer screen

Description automatically generated

I started first by importing all the files into the main interface using all of the algorithms in the file.

A screen shot of a computer screen

Description automatically generated

I started with password generation and created a new function that would run when the “generate” button is clicked. I did not pass any local variables into the function because all of the variables in the algorithm need to be global so they can be collected for the main interface. Using the input form, the password length box I can apply the input into the imported algorithm and then use the generated\_password variable to pass into the output box.

When I ran the first test of this feature, received an error when testing different inputs. After some further testing I found that the previous password was not being deleted before the new one was being added on top as shown here. So each time I cleared the box to restart the function the passwords would get progressively longer no matter how long the length was due to the previous password being added on top.

A screen shot of a computer code

Description automatically generated

To rectify this, I set all the variables that will be involved in the interface to empty. This will ensure that each time a function has run the variable will be cleared so that algorithms like the password generator will be able to be run repeatedly.

A screenshot of a login box

Description automatically generated

With the password generation working properly I could move on to strength testing.

A screen shot of a computer program

Description automatically generated

Much like password generation I created a new function that would act as the command for the “test strength” button. I then imported the strength check algorithm and used the number received from it to create a final\_score variable within the main menu. This enabled me to display the result as a tkinter message box which displays the result much more clearly. A screenshot of a login box

Description automatically generated

As seen above upon copying the generated password I clicked the “Test Strength” button, the message box displays the final score as very strong in the message box which would be the right score for that password.

A screenshot of a login box

Description automatically generated

A screenshot of a login box

Description automatically generated

Just to make sure the other were still being added up correctly I tested a couple of other options such as leaving it blank, which correctly displayed “invalid”, and “Password”, which correctly displayed weak so I am confident that this feature now works and I can move on to encryption and decryption.

A screenshot of a computer screen

Description automatically generated

For this being the decryption section of the algorithm, I would only need to use the decryption function because I want to use the encryption function to run automatically when saving the passwords to the CSV file.

A computer screen shot of a computer code

Description automatically generated

Using the same process as the other algorithms I can create a new variable within the main interface and then import the decrypt function and return the value into the output box. The only issue is that I cannot check that this works until I have some passwords saved using the encrypt and save algorithm. So for now I know that the function works I just have to trust that I imported it correctly.

## Milestone 7: Encryption and Saving

A blue screen with white text

Description automatically generated

I started by making sure that the encryption function was ready to be imported into the main interface by ensuring that all the variables were local and that it was using the right variables for the rest of the program. A screen shot of a computer program

Description automatically generated



A computer screen with white text

Description automatically generated



Using the variable names from the entry boxes in the encrypt and save section of the program, I made a function which opened a file in the application folder to store all of the inputs from the window.

A computer screen shot of white text

Description automatically generated

A grid of white squares

Description automatically generatedUsing the save button I made a function to get all of the inputs from the different boxes and then put them into the three columns in the saved accounts file.



A screenshot of a computer screen

Description automatically generatedThis proves that the saving part of the algorithm works as it should however, I noticed that when viewing the passwords in excel there were repeated characters that looked different from the shell when the encryption algorithm was running to test this theory, I copied the text from the password column in the Netflix row. The input into the password box was “Netflixpasswordtest1”. I then copied the text into the decryption box.

As seen on the right the output was far different from my original input, so I retested the encryption algorithm in the python shell using the same input of “Netflixpassword1”.

A screenshot of a computer screen

Description automatically generated As seen here the output from the same password using the same algorithm is completely different but it works when I copy this across to decrypt it the output comes out as “Netflixtestpassword1”. This means that the algorithm does work but something is going wrong when I copy across the encrypted passwords to the excel file format. After spending a long time looking into this issue if found that it was to do with the way that my encryption and decryption algorithms interact with the way that python interprets text known as the encoding. Because I’m using ROT-13 as my cypher with an extended character set some characters end up being rotated to ASCII values which translate to characters such as, “r” 🡪 “delete” and “s” 🡪 “escape”. Because python’s encoding is directly linked to ASCII it uses an encoding called “utf-8”

Decrypting the text from the shell worked so to fix the problem of the file encoding not working properly I can use the csv library to print the file into the shell this should automatically put it into the correct encoding for python meaning the decryption should work with the copied text. This is also where I can implement the password to keep the true encrypted passwords hidden.

A computer screen shot of a program

Description automatically generated

A screenshot of a computer program

Description automatically generated



By combining the master password check algorithm and attaching it to the highlighted section of code. This opens the file in the folder and copies the text into the shell provided that the final check = True. If it = False then as before an info box will pop up saying that the password is incorrect.

For this part I got some help from a classmate as well as using some information on a stack overflow discussion at [Stack Overflow - Where Developers Learn, Share, & Build Careers](https://stackoverflow.com/).

A screenshot of a computer

Description automatically generated

A computer screen shot of a blue screen

Description automatically generated



As seen above when I put my account name and password in correctly after clicking “view saved” the algorithm checks all the details and if all come out as correct the file will be fully printed out into the shell. The highlighted password is the result of a previous test where the input was “Password2”. If I decrypt this and it comes out as should. It will provide me with enough evidence to believe that this section of the program works, and I can call the project complete.

A screenshot of a computer screen

Description automatically generated

Since copying it into the decryption box worked and the result was as expected I felt happy to call the program complete and move onto the final tests.

# Testing to inform evaluation.

|  |  |  |  |
| --- | --- | --- | --- |
| Criteria | Measurement/ Evidence | Proof | Works as expected/ not expected |
| Storing multiple passwords | The user should be able to store at least 3 passwords in a clear locked file. | 1. >>?Tq5=eW[? 2. [NOV8B1°TE:A>= 3. 2Y^V=>?o?L>6p | Works as expected |
| Strong password randomiser button | The user should be able to generate at least 5 12 character passwords that would get a “Strong” rating when using the built in password checker. | 1. Or{cwc6Vl9%( (very strong) 2. .9ZPD1BMqkl4 (very strong) 3. ]E65I8tcSgZ3 (very strong) 4. @ghc&j6`cIG@ (very strong) 5. <&>pe^DH3a¬5 (very strong)   (see screenshot 2) | Works as expected |
| Encrypting and decrypting stored passwords | The user should be able to store 3 encrypted passwords in a locked file that do not resemble the original passwords. All 3 should then be successfully decrypted when the copy button is used. | 1. Website: Test 1   Account name: Test\_account\_name\_1  Password: Test\_Password\_1  (see screenshot 3 & 4)   1. Website: Test 2   Account name: Test\_account\_name\_2  Password: Test\_Password\_2  (see screenshot 5 & 6)   1. Website: Test 3   Account name: Test\_account\_name\_3  Password: Test\_Password\_3  (see screenshot 7 & 8) | Works as expected |
| Working copy button | The user presses the copy button which copies the encrypted password and decrypts it and save the decrypted text to the clipboard. 3 different passwords should be successfully copied and pasted to see if it matches the original password. | This feature was not included in the final program due to time restraints. I have got around this in testing by highlighting the text and using ctrl + c and ctrl + v. | Feature not included in the final program |
| Multiple accounts management | The user should be able to create at least 2 accounts that have 2 different databases of stored passwords and account details. | The code in the picture enables me to create a new username and attach a password which will be stored in a separate encrypted file, but I was not able to add this as a feature within the core program.  (see screenshot 9) | Feature partially complete but cannot be accessed using the program normally |
| Working password strength checker button | The password strength check should give a valid result for at least 5 passwords (valid result: a result that matches to the checklist of what makes a strong password). | As seen above in the generation all of the passwords generated came out as very strong which is the correct rating for the passwords. I also tested:   1. “password” (very weak) 2. “D1plod0cus” (not bad) 3. “D1plod0cus\_D!N0” (very strong)   (see screenshot 10, 11, 12) | Works as expected |
| Log in boxes work | The user should be able to type in the given details correctly to log into their specific account | I can access the locked contents by typing my account name and personal password.  (see screenshot 13) | Works as expected |
| Clear and concise design that is easy to interpret for all users | The user should be able to access all features of the program without given help | My stake holders were able to fully access the program and could navigate it completely within a short space of time. The only thing I wanted to include were images and design flourishes, but this was not possible due to the limitations of the libraries.  (see screenshot 14) | Feature partially complete |
| Passwords should be stored along with a website name and an account name | At least 3 full rows filled out complete with password, account name, and website | (See screenshot 15) | Works as expected |

A screenshot of a computer login

Description automatically generated

Screenshot 2

A blue background with white text

Description automatically generated

Screenshot 3

A screenshot of a computer screen

Description automatically generated

Screenshot 4

A blue background with white text

Description automatically generated

Screenshot 5

A screenshot of a login screen

Description automatically generated

Screenshot 6

A blue screen with white text

Description automatically generated

Screenshot 7

A screenshot of a computer screen

Description automatically generated

Screenshot 8

A screenshot of a login box

Description automatically generated

Screenshot 10

A login box with a blue circle and a blue circle with black text

Description automatically generated

Screenshot 11

A screenshot of a login box

Description automatically generated

Screenshot 12

### A screen shot of a computer code Description automatically generated

Screenshot 9: as seen below the code is a function that creates a new file to store a new user and master password. Although possible to create a new account the function is not yet linked to anything and does not have its own encrypted password file accessible through the interface.

## A screenshot of a computer screen Description automatically generated

Screenshot 13

## A screenshot of a login screen Description automatically generated

Screenshot 14

Screenshot 15

## A screenshot of a computer Description automatically generated

# Evaluation

Upon completion of the password and account management system project at this point in time, the vast majority of the requirements have been met that were first established in the analysis section. I have completed almost every goal I could complete with the given resources and time frame. As seen above in the testing to inform evaluation section of this document, some sections are still 2 goals that are partially complete and one goal that I just could not include with the resources I have had access to.

Looking at my final testing table I can see the only 3 goals left incomplete are: clear and concise interface with images to match the design, being able to manage the passwords of multiple accounts, and using a working copy button to copy and paste text to save time. Two of the three goals being the inclusion of images and a copy button were not possible to add to the program due to the fact that they required libraries that I did not have access to. The final one that I did not include was full access to creating multiple accounts. This was a key feature of the program that I had to leave out due to time constraints. The process of linking an account to its own password file that was separate from the current one proved to be far more complicated than I had originally thought therefore I did not have the time to create new interfaces required for this feature to work.

Apart from the copy button, which is the goal I have missed, I have fully included 5 goals (Storing multiple passwords, Strong password randomiser button, Encrypting and decrypting stored passwords, Working password strength checker button, Passwords should be stored along with a website name and an account name) and partially included 2 (Multiple accounts management, Clear and concise design that is easy to interpret for all users) out of 8 in total. Because of this I think it is safe to consider this as a successful program that has hit most of its goals.

If I was to repeat this project, I would give my self-access to more libraries and time. There are multiple ways of tackling the features that I could not include. One of the most prevalent being the exclusion of features from my interface. I did not have access to expanded libraries for example all my interfaces were made using the python tkinter library. This library does not support images without importing its expanded image library. If I could include this feature, I could have used the Pygame library which enables me to add images to a screen as a sprite however this would have made the code very complex forcing me to use object-oriented programming where I didn’t necessary need it. I could also have done this project on a home computer where I would have had full access to any library I could find. The same issue applies for creating a copy button. I needed additional libraries that I did not have access to so that I could access the clip board which would allow the user to copy the output with the click of a button.

## Stake Holder Input for Evaluation

When it came for the stake holders to test the program, I had to change the password from my personal one to “Password” so that they were able to access every part of the program that was available.

|  |  |  |
| --- | --- | --- |
| Questions | Answers Yes/No | Additional comments |
| I could navigate the program without any input from the developer | Yes (2)  No (1) | “It was not too clear which boxes were input and output boxes.” |
| I was able to generate my own password | Yes (3)  No (0) | N/A |
| I could test the strength of my passwords | Yes (3)  No (0) | N/A |
| I could save my passwords successfully | Yes (3)  No (0) | N/A |
| I was able to find my saved encrypted password when I entered the mater password (“password”) | Yes (3)  No (0) | N/A |
| I could decrypt my password and copy it to a browser window | Yes (2)  No (1) | The stake holder did not know how to copy and paste using keyboard shortcuts without external help |
| I could close the program without any input from the developer | Yes (3)  No (0) | N/A |
| I encountered a bug when testing the program | Yes (0)  No (3) | N/A |
| Any further comments |  | 1. “The program works well with its functions, and I like the simplicity of it.” 2. “Every part of the program worked it would be nice if there was a few quality-of-life features like a copy button or automatically removing the outputs when something else is being generated” 3. “I liked the number of features included in the program, but the interface was not very interesting” |

## What I Could Improve

Based on the inputs from the stake holders, they are very happy with the features included in the program but some aspects such as ease of access and a few quality-of-life features. I think the first change I would make is making the output boxes clearer as well as making them un-editable so the outputs can’t be accidently changed by the user. I think it would also be useful to have the boxes automatically clear when a new password is decrypted or generated and replaced by the new output in one go. This is not a major feature, but I think it would make a substantial difference to how user friendly the program is. I think a final small additional feature would be giving a display message when a password has been successfully encrypted and saved. As mentioned before the copy button, access to images and the capability to create multiple accounts were all planned features that I could not complete but all 3 would greatly improve the user experience.

In terms for larger scale future plans for the project it would be fantastic to include some of the features that are key to apps like Nord Pass and Dash Lane where they can automatically fill in the log in boxes to a website log in portal. I did not have time to do research or look further into this feature since its complexity was far beyond my current programming capability. But it would be nice to come back to in time. I would also like to use a different interface library that would enable me to add features like pictures and change the way that buttons look. Using a different library will allow me to give the program a much more professional look.

As well as these improvements I also think that it will be more practical to store the saved passwords into an online database. This way you have much more control with how the data is accessed making it more secure.

## What I have learned

Throughout my time completing this project I have come across many challenges stemming from different problems from a lack of available research due to the inspired programs being locked behind a paywall. Or problems with how encoding affects how text and ASCII values interact with each other like the errors in pages 51 and 52. In all instances I have learned how to change my approach when I encounter an error. Even if that involves changing the initial goal to meet the criteria. An example would be the encoding

# Bibliography

Stack Overflow, URL: [dictionary - How can I use pickle to save a dict (or any other Python object)? - Stack Overflow](https://stackoverflow.com/questions/11218477/how-can-i-use-pickle-to-save-a-dict-or-any-other-python-object), Author of Response: CryptoFool, used October 2023

Geeks for Geeks, URL: [Python Tkinter Tutorial - GeeksforGeeks](https://www.geeksforgeeks.org/python-tkinter-tutorial/?ref=lbp) , Author of article: nikhilaggarwal3, used October 2023

Micheal Parker (classmate), help with encoding when saving to a file, October 2023

W3 Schools, URL: [Python Tutorial (w3schools.com)](https://www.w3schools.com/python/default.asp), used the tutorial on managing lists, Author of page: Unknown, September 2023

# Appendices

## Code dump

### **Encryption and decryption**

# encrypt and decrypt with ROT13 using ord() and chr() with object oriented

# defining and naming the separate lists

num\_list = []

encrypt\_list = []

decrypt\_list = []

num\_list2 = []

# the main function of this algorithm where the user inputs the message to encrypt and decrypt

def main():

decrypted\_password = input("input a message to encrypt: ")

result = encrypt(message1)

print(result)

encrypted\_password = input("input a message to decrypt: ")

result = decrypt(message2)

print(result)

# encryption function

def encrypt(new\_password):

# clear both lists

for c in new\_password:

# convert to ascii and add 13 to every character in message1

y = ord(c) + 13

num\_list.append(y)

for i in num\_list:

# convert every number in num\_list back into characters

z = chr(i)

encrypt\_list.append(z)

# convert to a string by joining each letter in the list with no space

a = "".join(encrypt\_list)

num\_list.clear()

encrypt\_list.clear()

return a

# decryption function

def decrypt(encrypted\_password):

# clear both of the lists

for c in encrypted\_password:

# convert each character in message2 into ASCII then - 13 from the value

t = ord(c) - 13

num\_list2.append(t)

for i in num\_list2:

# convert the value back into characters

u = chr(i)

decrypt\_list.append(u)

# turn the decrypt\_list into a string by joining them with no spaces

b = "".join(decrypt\_list)

num\_list2.clear()

decrypt\_list.clear()

return b

### **main menu interface**

# interface (main menu)

import tkinter as tk

from tkinter import messagebox

import csv

import password\_generation

import strength\_check

import encryption\_and\_decryption

import save\_to\_file

import master\_password\_storage

# set all variables to null

pswd\_length = ""

generated\_password = ""

password = ""

encrypted\_password = ""

decrypted\_password = ""

website\_name = ""

account\_name = ""

new\_password = ""

# define the main window and window size

win = tk.Tk()

frame = tk.Frame(win)

win.geometry("1000x400")

# Main menu title

main\_title = tk.Label(win, text="KeyRing", font=('Arial bold', 80))

main\_title.pack()

# Password generation title

password\_generation\_title = tk.Label(win, text="Password Generation", font=('Arial bold', 20))

password\_generation\_title.place(x=30, y=140)

# Password length subtitle

password\_length\_title = tk.Label(win, text="Password Length", font=('Arial', 20))

password\_length\_title.place(x=30, y=180)

# Password length entry

password\_length = tk.Entry(win, textvariable= pswd\_length, font=('Arial',10))

password\_length.place(x=30, y=220)

# Function to generate the password

def generate\_click():

# get the password length from the password\_length input box

pswd\_length = int(password\_length.get())

# run the generate function from the password\_generation file

generated\_password = password\_generation.generate(pswd\_length)

# insert the output text into the output box

password\_output.insert(0, generated\_password)

# Button that generates password

generate\_button = tk.Button(win, text=('Generate'), font=('Arial', 10), command= generate\_click)

generate\_button.place(x=30, y=245)

# Output box where the password is displayed

password\_output = tk.Entry(win, textvariable= generated\_password, font=('Arial',10))

password\_output.place(x=30, y=280)

# Strength check subtitle

strength\_check\_title = tk.Label(win, text=('Strength Test'), font=('Arial',20))

strength\_check\_title.place(x=30, y=310)

# Strength check input box

strength\_check\_input = tk.Entry(win, textvariable= password, font=('Arial',10))

strength\_check\_input.place(x=30, y=345)

# create a function for the strength check button to run the find score function in the strength\_check file

def test\_strength():

password = strength\_check\_input.get()

score\_result = strength\_check.find\_score(password)

# get the score and assign a rating to the value

if score\_result == 0:

final\_score = 'invalid'

if score\_result == 1:

final\_score = 'very weak'

if score\_result == 2:

final\_score = 'weak'

if score\_result == 3:

final\_score = 'not bad'

if score\_result == 4:

final\_score = 'strong'

if score\_result == 5:

final\_score = 'very strong'

# output the rating in an info message box

tk.messagebox.showinfo('password score', final\_score)

# Strength check button

strength\_check\_button = tk.Button(win, text=('Test Strength'), command= test\_strength)

strength\_check\_button.place(x=30, y=370)

# Decryption subtitle

decryption\_title = tk.Label(win, text=('Decrypt Password'), font=('Arial bold', 20))

decryption\_title.place(x=380, y=140)

# Enter encrypted password

enter\_encrypted\_title= tk.Label(win, text=('Enter encrypted password:'), font=('Arial', 10))

enter\_encrypted\_title.place(x=380, y=180)

# Decryption entry box

decryption\_input = tk.Entry(win, textvariable= encrypted\_password, font=('Arial', 10))

decryption\_input.place(x=385, y=210)

# Decrypted output box

decryption\_output = tk.Entry(win, textvariable= decrypted\_password, font=('Arial', 10))

decryption\_output.place(x=385, y=280)

# make a decrypt function that can be run as a command on the button

def decrypt():

# get the input from the decryption input box

encrypted\_password = decryption\_input.get()

# run the decrypt function using the text from the box

decrypted\_password = encryption\_and\_decryption.decrypt(encrypted\_password)

# insert the decrypted text into the output box

decryption\_output.insert(0, decrypted\_password)

# Decryption Button

decryption\_button = tk.Button(win, text=('Decrypt'), font=('Arial', 10), command= decrypt)

decryption\_button.place(x=385, y=240)

# encrypt and save

save\_title = tk.Label(win, text=('Encrypt and Save'), font=('Arial bold', 20))

save\_title.place(x=700, y=140)

# website name

website\_title = tk.Label(win, text=('Website Name:'), font=('Arial', 10))

website\_title.place(x=700, y=170)

# website entry

website\_entry = tk.Entry(win, textvariable= website\_name, font=('Arial', 10))

website\_entry.place(x=700, y=190)

# account name title

account\_name\_title = tk.Label(win, text=('Account Name:'), font=('Arial', 10))

account\_name\_title.place(x=700, y=210)

# account name entry box

account\_name\_entry = tk.Entry(win, textvariable= account\_name, font=('Arial', 10))

account\_name\_entry.place(x=700, y=230)

# Password title

password\_title = tk.Label(win, text=('Password:'), font=('Arial', 10))

password\_title.place(x=700, y=250)

# Password entry box

password\_entry = tk.Entry(win, textvariable= new\_password, font=('Arial', 10), show='\*')

# the "show='\*'" part of the line above displays the text in the box as \*s

password\_entry.place(x=700, y=270)

def save():

website\_name = website\_entry.get()

account\_name = account\_name\_entry.get()

new\_password = password\_entry.get()

encrypted\_new\_password = encryption\_and\_decryption.encrypt(new\_password)

print(encrypted\_new\_password)

save\_to\_file.save\_details(website\_name, account\_name, encrypted\_new\_password)

# Save Button that runs the save function

save\_button = tk.Button(win, text=('Save'), font=('Arial', 10), command= save)

save\_button.place(x=700, y=300)

def view\_saved():

master\_password\_storage.view\_passwords()

# view saved passwords button

view\_saved = tk.Button(win, text=('View Saved'), font=('Arial', 10), command= view\_saved)

view\_saved.place(x=700, y=330)

win.mainloop()

### **master password storage**

# master password

import \_pickle as cPickle

import hashlib

import tkinter as tk

from tkinter import messagebox

import subprocess

account\_check = ""

password\_check = ""

def view\_passwords():

win2 = tk.Tk()

frame = tk.Frame(win2)

win2.geometry("300x150")

# account entry title

account\_entry\_title = tk.Label(win2, text=('Account Name:'), font=('Arial', 10))

account\_entry\_title.pack()

# account entry box

account\_entry\_box =tk.Entry(win2, textvariable= account\_check, font=('Arial', 10))

account\_entry\_box.pack()

# password entry title

password\_entry\_title = tk.Label(win2, text=('Password:'), font=('Arial', 10))

password\_entry\_title.pack()

# password entry box

password\_entry\_box = tk.Entry(win2, textvariable= password\_check, font=('Arial', 10), show='\*')

password\_entry\_box.pack()

def check\_password():

# open the 2 pickle files and read the contents of them

with open('account\_file.pkl', 'rb') as f:

account\_name\_loaded = cPickle.load(f)

print(account\_name\_loaded)

with open('master\_password\_file.pkl', 'rb') as f:

master\_password\_loaded = cPickle.load(f)

print(master\_password\_loaded)

account\_check = account\_entry\_box.get()

# Check if the Account name entered in the box is the same as the one in the pickle file

if account\_check == account\_name\_loaded:

account\_check = True

else:

account\_check = False

# Hash the user input in the entry box using sha256

password\_check = password\_entry\_box.get()

hash\_check = hashlib.sha256(password\_check.encode())

hex\_check = hash\_check.hexdigest()

# Check if the hashed passwords match

if hex\_check == master\_password\_loaded:

password\_check = True

else:

password\_check = False

# Final Checks and print the results in the shell

if account\_check == True:

print("account correct")

else:

print("account incorrect")

if password\_check == True:

print("password correct")

else:

print("password incorrect")

if account\_check and password\_check == True:

final\_check = True

else:

final\_check = False

if final\_check == False:

tk.messagebox.showinfo('password score', 'incorrect')

if final\_check == True:

print("Correct")

file = open("accounts\_file.csv", mode = 'r', encoding='utf-8')

print(file.read())

file.close()

# check password button

check\_password\_button = tk.Button(win2, text=('Enter'), font=('Arial', 10), command= check\_password)

check\_password\_button.pack()

def decryption():

win2 = tk.Tk()

frame = tk.Frame(win2)

win2.geometry("300x150")

# account entry title

account\_entry\_title = tk.Label(win2, text=('Account Name:'), font=('Arial', 10))

account\_entry\_title.pack()

# account entry box

account\_entry\_box =tk.Entry(win2, textvariable= account\_check, font=('Arial', 10))

account\_entry\_box.pack()

# password entry title

password\_entry\_title = tk.Label(win2, text=('Password:'), font=('Arial', 10))

password\_entry\_title.pack()

# password entry box

password\_entry\_box = tk.Entry(win2, textvariable= password\_check, font=('Arial', 10), show='\*')

password\_entry\_box.pack()

def decryption\_check():

with open('account\_file.pkl', 'rb') as f:

account\_name\_loaded = cPickle.load(f)

print(account\_name\_loaded)

with open('master\_password\_file.pkl', 'rb') as f:

master\_password\_loaded = cPickle.load(f)

print(master\_password\_loaded)

account\_check = account\_entry\_box.get()

if account\_check == account\_name\_loaded:

account\_check = True

else:

account\_check = False

password\_check = password\_entry\_box.get()

hash\_check = hashlib.sha256(password\_check.encode())

hex\_check = hash\_check.hexdigest()

if hex\_check == master\_password\_loaded:

password\_check = True

else:

password\_check = False

if account\_check == True:

print("account correct")

else:

print("account incorrect")

if password\_check == True:

print("password correct")

else:

print("password incorrect")

if account\_check and password\_check == True:

final\_check = True

else:

final\_check = False

return final\_check

# check password button

check\_password\_button = tk.Button(win2, text=('Enter'), font=('Arial', 10), command= check\_password)

check\_password\_button.pack()

# function to encrypt and store a master password and aan account name in seperate files

def new\_user():

# get the desired user name and password from the user

master\_password = input("new password: ")

account\_name = input("account name: ")

# hash the master password using the SHA256 algorithm

hash\_object = hashlib.sha256(master\_password.encode())

hex\_dig = hash\_object.hexdigest()

# open the pickle files and store the hashed password and account name

with open('account\_file.pkl', 'wb') as f:

cPickle.dump(account\_name, f)

with open('master\_password\_file.pkl', 'wb') as f:

cPickle.dump(hex\_dig, f)

# return the hashed password for later

return(hex\_dig)

### **Password generation**

import random

# store all of the lists for the character types

lower\_letter = ['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']

upper\_letter = ['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']

number = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '0']

symbol = ['!','£','$','%','^','&','\*','(',')','-','\_','+','=','{','}','[',']',':',';','@','#','~','<',',','>','.','/','?','¬','`']

characters = ['lower\_letter', 'upper\_letter', 'number', 'symbol']

character\_list = []

pswd\_list = []

def main():

pswd\_length = 0

pswd\_length = int(input("enter the desired length of your password: "))

password = generate(pswd\_length)

print(password)

# define the generate function

def generate(pswd\_length):

# generate a list with a random combination of character types

for x in range(pswd\_length):

x = random.choice(characters)

character\_list.append(x)

# checks to make sure that at least one of each character type is included in the final password

for x in range(4):

# if lower case letter is not in the list delete the first item and add a lower case letter on the end

if 'lower\_letter' not in character\_list:

del character\_list[0]

character\_list.append('lower\_letter')

# if upper case letter is not in the list delete the first item and add an upper case letter on the end

elif 'upper\_letter' not in character\_list:

del character\_list[0]

character\_list.append('upper\_letter')

# if a number is not in the list delete the first item and add a number on the end

elif 'number' not in character\_list:

del character\_list[0]

character\_list.append('number')

# if a symbol is not in the list delete the first item and add a symbol on the end

elif 'symbol' not in character\_list:

del character\_list[0]

character\_list.append('symbol')

# for loop that checks each item in the character\_list if the item in the lists matches one of the stored

# character types, take a random item from the specific list and add it to the pswd\_list.

for x in character\_list:

if x == 'lower\_letter':

a = random.choice(lower\_letter)

pswd\_list.append(a)

elif x == 'upper\_letter':

a = random.choice(upper\_letter)

pswd\_list.append(a)

elif x == 'number':

a = random.choice(number)

pswd\_list.append(a)

elif x == 'symbol':

a = random.choice(symbol)

pswd\_list.append(a)

# join the password list with no spaces to make it into a string

complete\_pswd = "".join(pswd\_list)

print(complete\_pswd)

character\_list.clear()

pswd\_list.clear()

return complete\_pswd

### **Save to file**

import csv

#take all the variables from the input boxes in the main file and add them to the function

def save\_details(website\_name, account\_name, encrypted\_new\_password):

save\_list = []

save\_list.append(website\_name)

save\_list.append(account\_name)

save\_list.append(encrypted\_new\_password)

# open the accounts\_file and use utf-8 encoding

with open('accounts\_file.csv', mode='a', encoding='utf-8') as accounts\_file:

# set the rules for writing each row and seperate each item with a comma

accounts\_writer = csv.writer(accounts\_file, delimiter=',')

# add the list to the current row then go down to the next row

accounts\_writer.writerow(save\_list)

save\_list.clear()

### **Password strength checker**

# Strength checker

# defining the character types and setting the variables to 0

lower\_letter = ['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']

upper\_letter = ['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']

number = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '0']

symbol = ['!','£','$','%','^','&','\*','(',')','-','\_','+','=','{','}','[',']',':',';','@','#','~','<',',','>','.','/','?','¬','`']

characters = ['lower\_letter', 'upper\_letter', 'number', 'symbol']

strength\_list = []

score = 0

length = 0

def assign\_rating():

# input the password

# run the function to find the score

score = find\_score(password)

# assign a rating to each score

# if the score is 0, nothing has been entered, therefore it is invalid

if score == 0:

print('invalid')

if score == 1:

print('very weak')

if score == 2:

print('weak')

if score == 3:

print('not bad')

if score == 4:

print('strong')

if score == 5:

print('very strong')

def find\_score(password):

score = 0

length = 0

# change the password into a list

pswd\_list = list(password)

# make a new list called strength list and add the all the character types

# for each character in the password list

for x in pswd\_list:

if x in lower\_letter:

strength\_list.append("lower\_letter")

elif x in upper\_letter:

strength\_list.append("upper\_letter")

elif x in number:

strength\_list.append("number")

elif x in symbol:

strength\_list.append("symbol")

# check the list to see if the specific charcter type is in it

# if it is add one to the score

if "lower\_letter" in strength\_list:

score = score + 1

if 'upper\_letter' in strength\_list:

score = score + 1

if 'number' in strength\_list:

score = score + 1

if 'symbol' in strength\_list:

score = score + 1

# add one to the score if the length is 12 characters or more

for x in strength\_list:

length = length+1

if length > 11:

score = score + 1

strength\_list.clear()

return score