Passwortmanager

BETRIEBSSYSTEME UND RECHNERNETZE

Dozent: Christian Baun

MOHAMMAD AHMADI	1349223
SAMUEL JALLOW-SZUCKI	1354373
LILY SCHRETTER	1360118

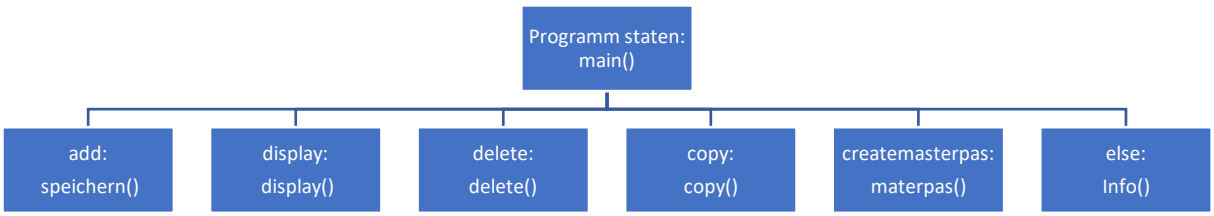
Inhaltsverzeichnis

1. Einleitung

Ein „Passwort-Manager“ ist immer vorteilhaft. Sei es für private oder unternehmerische Zwecke. Ein Programm, um die Daten zu sichern zu verstauen und flexibel darauf zugreifen zu können ist in unserem Digitalzeitalter von großer Wichtigkeit. In der folgenden Dokumentation unseres Programms zum „Passwort-Manager“ wird detailliert auf die Struktur, Prozesse, Befehle und Funktionen des Programms eingegangen. Unter anderem wird eine Erklärung unseres Vorgehens angegeben und wie die Anforderungen gelöst wurden. Ein Feedback und weitere Schwierigkeiten oder Herausforderungen, die beim Erstellen des Passwortmanagers entstanden sind, werden auch ausführlich beschrieben.

2. Code Architektur/Code Struktur

Um den Code übersichtlich zu gestalten, wurde das Programm in drei verschiedenen Python-Files realisiert, wobei die Main-File die Anfragen des Nutzers prüft und anschließend die entsprechenden Funktionen aufruft. Diese werden in den Files „Module_direkt“ und „Module_indirekt“ programmiert. Das heißt in der „Main“ werden alle Funktionen von „module_direkt“ importiert und in diese werden alle Funktionen von „module_indirekt“ importiert.
Die „Main“ besteht aus einer Funktion „main()“, in der durch If-Abfragen geschaut wird, welcher Befehl vom Benutzer im Terminal eingegeben wurde. Für jeden Befehl ist eine Funktion zuständig, die von „module_direkt“ importiert wurde. Durch das Diagramm soll verdeutlicht werden, welche Befehle es gibt und welche Funktionen für die einzelnen Befehle zuständig sind.



3. Eigene Kommandozeilenargumente

Zum Sarten des Programms aus dem Terminal heraus muss es einen eigenen Kommandozeilen Befehl geben, damit man nicht immer den kompletten im Pfad im Terminal eingeben muss. Hierzu gibt es einen weiteren „setup“- File, in der man aus dem Modul „setuptools“ „setup“ importiert. Nun wird ein eigenes Modul erstellt, in dem man den Befehlsnamen „pasman“ vergibt, der die Aufgabe hat aus dem Programm die „Main“- File auszuführen. Bevor man aber den Befehl nutzen kann, muss er zunächst im Terminal installiert werden.
Für das Programm müssen außerdem Argumente erstellt werden, damit das Programm im Terminal starten kann. Hierzu muss das Modul „Argparse“ in „module_indirekt“ importiert werden, damit positionale und optionale Argumente erstellt werden können. Zum Starten des Programms ist immer ein Befehl/Argument notwendig, der bestimmt, was getan werden muss. Daher muss für das Programm das positionale Argument „befehl“ erstellt werden, das zum Beispiel „add“, „copy“, „display“, „delete“ oder „createmasterpas“ sein kann.
Allerdings benötigen wir für das Passwortmanager je nach Befehl weitere Argumente, die optional sein sollten. Da für das Passwortmanager ein Titel, Username und ein Passwort notwendig ist, müssen drei weitere optionale Argumente hinzugefügt werden. Die Argumente müssen optional sein, da z.B für den Befehl „add“ (speichern) ein Titel, Username und Passwort gebraucht wird, aber für den Befehl „delete“ wird nur der Titel benötigt. Mit dem Hinzufügen eines Arguments kann man auch eine Helpfunktion einbauen, in der man dann einen Text formulieren kann, der dem Benutzer die Funktion des Arguments verdeutlichen soll.
Für das Passwortmanager müssen also optionale Argumente für den Titel, Username und das Passwort hinzugefügt werden.
Nun soll aber das Passwort automatisch generiert werden. Das heißt der Benutzer soll für das Argument „generatepassword“ keinen Inhalt eingeben, da durch das Eingeben des Arguments automatisch ein Passwort erstellt werden soll.
Wenn man allerdings nur „- -generatepassword“ eingibt, kann das Programm nicht durchgeführt werden, da das Argument einen Inhalt benötigt. Das Problem wird gelöst, indem man die Actionfunktion beim Hinzufügen des Arguments nutzt, um das Argument immer auf True zu setzen. So wird nach der Eingabe des Arguments kein Inhalt mehr benötigt und das Ergebnis ist immer True. Mithilfe einer If - Abfrage kann man jetzt in „module_direkt“ File abfragen: Wenn das Argument „generatepassword“ True ist (was durch die Actionfunktion immer auf True gesetzt wurde), dann entspricht das Argument „generatepassword“ der Funktion „generatepassword()“. Die Funktion „generatepassword()“ gibt ein automatisch generiertes Passwort wieder.

generatepassword()
Der Benutzer wird am Anfang aufgefordert einzugeben wie lang das Passwort sein soll und aus welchen Zeichen das Passwort bestehen soll. Mit einer If – Abfrage wird danach überprüft für welche Zeichenkombination sich der Benutzer entschieden hat.
Mit einer For-Schleife, der so oft durchlaufen wird wie die Länge des Passworts, kann man nun ein Passwort generieren. Bei jedem Durchlauf wird hierzu ein Zeichen aus der Zeichenkombination, die der Benutzer zuvor ausgewählt hat, ein zufälliges Zeichen dem

leeren String „Passwort“ hinzugefügt. Das zufällige Auswählen eines Zeichens bei jedem Durchlauf erfolgt durch das Modul „random“.
Zum Schluss wird das generierte Passwort zurückgegeben.

4. Erklärung der einzelnen Befehle

Im Folgenden werden wir die Funktionsweise der einzelnen Befehle erklären. Die Befehle greifen immer auf eine Funktion der „module_direkt“ File zu, die wiederrum auf mehrere Funktionen der „module_indirekt“ zugreifen kann.

4.1 Befehl: „add“

Durch den Befehl „add“ wird auf die Funktion **„speichern()“** in „module_direkt“ zugegriffen. Die Hauptaufgabe der Funktion ist es die eingegeben Einträge in eine Textdatei zu speichern. In der Funktion werden zunächst drei Textdateien im Modus „a“ geöffnet, in die man etwas reinschreiben kann. Durch den Modus „a“ kann man Text an eventuell bestehendem Text anhängen.
Das generierte Passwort sollte aber nicht in originaler Form in der Textdatei stehen, da jeder Zugriff auf die Datei hat. Demnach muss das Passwort also verschlüsselt in der Textdatei stehen. Zum Verschlüsseln des Passworts greift die Funktion auf die Funktion **„enc()“** von „module_indirekt“ zu. Für diese Funktion wurde das Modul „base64“ importiert, mit dem man Passwörter verschlüsseln kann, aber diese auch wieder entschlüsseln kann. Da die Passwörter durch einen anderen Befehl auch in Originalform angezeigt werden sollten, ist die Option der Entschlüsselung der Passwörter unbedingt notwendig. In der Funktion „enc()“ wird das zuvor generierte Passwort encoded beziehungsweise verschlüsselt. Der Rückgabewert ist das in der Variable „encpas“ gespeicherte verschlüsselte Passwort.
Die Eingaben können jetzt in die geöffneten und dazugehörigen Textdateien mit der Methode „write“ gespeichert werden. In die Passwortdatei „pa.txt“ wird demnach nicht das originale Passwort geschrieben, sondern das verschlüsselte Passwort „passwortenc“. Zum Schluss wird das Passwort mit der Funktion **„kopieren()“** kopiert. Für das Kopieren ist einmal das Modul „pyperclip“ notwendig, womit man Sachen in der Zwischenablage kopieren kann, durch das Modul „time“ kann man bestimmen wie lange das Passwort gespeichert werden soll und durch das Modul „os“ kann man ein Kindprozess erstellen, in der etwas unabhängig vom Elternprozess stattfindet. In der Funktion wird zunächst das Passwort mit „pyperclip.copy“ in der Zwischenablage gespeichert. Danach wird durch „os“ „fork“ aufgerufen, wodurch eine Kopie vom aktuellen Prozess erstellt wird und als Kindprozess unabhängig vom Elternprozess weiterläuft. Durch die If – Abfrage wird der Kindprozess auf True gesetzt, da der Kindprozess den Rückgabewert 0 hat. In dieser Abfrage soll dann zunächst die Zeit für 15 Sekunden durch „time.sleep“ einfrieren und es soll nichts passieren. Nach diesen 15 Sekunden kann man mit „pyperclip.paste“ wieder abfragen, ob in der Zwischenablage das Passwort kopiert ist. Sollte dies der Fall sein, dann soll das Programm einen leeren String in die Zwischenablage kopieren. Bei dieser Funktion ist das Erzeugen eines Kindprozesses notwendig, da der Benutzer eventuell in diesen 15 Sekunden im Terminal weiterarbeiten möchte. Durch das nicht erzeugen eines Kindprozesses müsste der Benutzer die 15 Sekunden abwarten, bevor er am Programm weiterarbeiten kann.

4.2 Befehl: „display“

Durch den Befehl „display“ wird auf die Funktion **„display“** in „module_direkt“ zugegriffen. Durch das Eingeben des Befehls soll zunächst nach einem Masterpasswort gefragt werden und nach der richtigen Eingabe des Masterpassworts sollen alle Einträge in einer Tabelle angezeigt werden.
Die Funktion besteht aus einer Funktion **„Masterpw_Eingabe()“,** die von „module_indirekt“ importiert wird. In der Funktion wird wiederrum erst einmal durch eine If – Abfrage kontrolliert, ob in der Textdatei „mp.txt“, in der das Masterpasswort gespeichert ist, etwas drin steht. Die Kontrolle erfolgt durch die Funktion **„checkfile(filename)“**, die auch in „module_indirekt“ enthalten ist. In dieser Funktion wird in einer If - Abfrage durch „os.stat(filename)“ zunächst „filename“ aufgerufen und danach wird mit „st_size“ kontrolliert, ob die Größe der Datei größer als Null ist. Sollte das Ergebnis größer als Null sein, dann steht in „filename“ etwas drin und es soll demnach „True“ zurückgegeben werden. Andernfalls ist die Datei noch leer und das wiederrum heißt, dass kein Masterpasswort erstellt wurde. In diesem Fall wird durch eine Printfunktion der Hinweis gegeben, dass ein Masterpasswort mit dem Befehl „createmasterpas“ erstellt werden muss. Wenn aber in der Textdatei „mp.txt“ etwas vorhanden ist, dann wird die Textdatei mit dem Modus „r“ geöffnet und die erste Zeile in der Datei wird als Variable „maspw_txt_lesen“ abgespeichert. Nun wird der Benutzer aufgefordert das Masterpasswort einzugeben. Damit das eingegebene Passwort nicht in Klartext angezeigt wird, wird die Funktion **„hide_pasw()“** aufgerufen. Für die Funktion wird „stdiomask“ importiert, wodurch die Eingabe des Benutzers nicht klar angezeigt wird. „Prompt“ ist der Text zur Abfrage des Masterpassworts und „mask“ ist dem Zeichen „*“ gleichgesetzt, sodass jedes eingegebene Zeichen durch das Zeichen „*“ ersetzt wird.
Im nächsten Schritt wird die Eingabe des Benutzers mit dem Masterpasswort, der in der Variable „maspw_txt_lesen“ eingespeichert ist durch die Funktion **„masterpw_vergleich()“** verglichen. Für diese Funktion ist das Modul „bcrypt“ nötig, da das Masterpasswort auch durch „bcrypt“ verschlüsselt in der Textdatei eingespeichert wird. Mit der Methode „bcrypt.checkpw()“ kann das eingegebene Passwort durch den Benutzer mit dem gehashten/verschlüsselten Passwort aus der Textdatei „mp.txt“ verglichen werden. Mit einer If – Abfrage wird „True“ zurückgegeben, wenn der Benutzer das Passwort richtig eingibt und in allen anderen Fällen wird dem Benutzer mitgeteilt, dass das Passwort falsch eingegeben wurde.
Wenn nun in der „display()“ Funktion die If – Abfrage „True“ ist, dann wird mit der Printfunktion die Tabelle erstellt und es soll in einer For-Schleife aus den Textdateien „ti.txt“, „us.txt“ und „pa.txt“ die gleichen Zeilen ausgegeben werden. Die For-Schleife wird so oft durchlaufen wie die Anzahl der Zeilen in der Titeltextdatei. Die Ermittlung der Zeilenanzahl erfolgt mit der Funktion **„anzahl_zeilen()“,** in der auch mit einer For-Schleife die Anzahl der Zeilen als Zahl zurückgegeben wird. Das Passwort muss aber zuvor mit der Funktion **„dec()“** entschlüsselt werden. Die Funktion „dec()“ wird verwendet, um ein bereits verschlüsseltes Passwort zu entschlüsseln. Die Entschlüsselung des verschlüsselten Passworts aus der Passworttextdatei muss aber dabei von Index an der Stelle 2 bis zum Vorletzten Index passieren. Anstelle des verschlüsselten Passworts wird dann logischerweise das entschlüsselte Passwort ausgegeben.

4.3 Befehl: „createmasterpas“

Mit dem Befehl „createmasterpas“ kann der Benutzer ein Masterpasswort und eine Sicherheitsfrage erstellen bzw. ändern. Hierfür ist die Funktion **„masterpas“** in „module_direkt“ zuständig. Dazu muss zunächst überprüft werden, ob es bereits ein Masterpasswort gibt. Dies funktioniert mit der bereits erklärten Funktion **„checkfile()“.** Sollte die Textdatei leer sein, dann wird der Benutzer aufgefordert ein Masterpasswort einzugeben. Zuvor wird er allerdings mit der Funktion **„passwort_bedingungen()“** darüber informiert, welche Bedingungen erfüllt sein müssen, damit das Passwort sicher ist. Nach der Passworteingabe des Benutzers wird dem Benutzer seine Passwortstärke mit der Funktion **„passwort_sicherheit()“** angezeigt. Ist das Passwort kleiner als mindestens acht Zeichen, erhält es keinen Stern. Ist es länger als sieben Zeichen erhält das Passwort einen Stern. Einen weiteren Stern erhält das Passwort, wenn es mindestens einen Großbuchstaben enthält. Den letzten Stern bekommt das Passwort, wenn mindestens eine Zahl vorhanden ist. Drei Sterne sind die stärkste Sicherheit und wenn es weniger als drei Sterne hat, ist die Passwortstärke unsicher.
Nachdem das Masterpasswort erstellt wurde, kann der Benutzer mit der Funktion **„sicherheits_frage_erstellen()“** eine Sicherheitsfrage und eine Antwort dazu selbst formulieren, die in unterschiedlichen Textdateien gespeichert werden. Die Abfrage der Sicherheitsfrage erfolgt nur dann, wenn der Benutzer das Masterpasswort ändern möchten. Dies ist eine weitere Zusatzfunktion für die wir uns entschieden haben, da so die Sicherheit des Programms durch eine zusätzliche Schicht geschützt wird.
Sollte aber bereits ein Masterpasswort bestehen, so muss das Masterpasswort geändert werden. Dazu muss der Benutzer zunächst das alte Masterpasswort eingeben. Mit der Funktion **„masterpw_vergleich“,** die bereits erklärt wurde, erfolgt die Kontrolle. Bei der richtigen Eingabe des alten Masterpassworts, wird mit der Funktion **„sicherheit_frage()“** die Sicherheitsabfrage gestellt und überprüft. Die Eingabe des Benutzers wird dabei mit der Antwort aus der Antworttextdatei überprüft. Sollte die Antwort auch richtig sein, wird das alte Masterpasswort aus der Textdatei gelöscht und der Benutzer kann ein neues Masterpasswort erstellen. Mit den Funktionen **„passwort_bedingungen()“** und **„passwort_sicherheit()“** kann der Benutzer noch einmal die Bedingungen für ein starkes Passwort sehen. Das neue Masterpasswort wird demnach verschlüsselt in die Textdatei gespeichert. Die alte Sicherheitsabfrage wird mit der Funktion **„sicherheitfrage_neu()“** gelöscht und der Benutzer kann eine neue Sicherheitsabfrage mit einer Antwort selbst formulieren.

4.4 Befehl: „delete“

Für den Befehl „delete“ ist die Funktion **„delete()“** von „module_direkt“ zuständig. Mithilfe der Funktion **„delete()“** können Einträge gelöscht werden. Wie bei jeder Aktion wird hier zunächst die Funktion **„Masterpw_Eingabe()“** aufgerufen, um die Berechtigung des Nutzers zu prüfen. Sobald das richtige Masterpasswort eingegeben wurde, folgt eine weitere Bedingung, die durch die Funktion **„suche_titel()“** prüft, ob überhaupt der Titel vorhanden ist.
Die Funktion „suche_titel() sucht in einer Titeltextdatei durch eine For-Schleife nach dem eingegeben Titel und speichert danach die Zeilennummer des Titels in der Variable „zeilennum“ ab.

Von der Zeilennummer muss in der nächsten Zeile noch 1 abgezogen werden, da bei Index 0 begonnen wird. Ist ein Titel vorhanden, wird der Nutzer gefragt, ob der Eintrag wirklich gelöscht werden soll. Die Antwort muss aktiv eingetippt werden, wobei das Wort „ja“ sowohl klein als auch groß geschrieben zum Löschen des Eintrags führt. Wenn das Programm das entsprechende Wort nicht über die Tastatur eingegeben bekommt, wird der Eintrag nicht gelöscht, was dem Nutzer durch ein print-Statement vermittelt wird. Über das Schlüsselwort „del“ werden die Zeilen mit den Werten Titel, Benutzername und Passwort an der entsprechenden Stelle („zeilennum“) aus den einzelnen Textdateien gelöscht. Anschließend werden diese Textdateien zum Schreiben geöffnet und jeweils einzelnen Variablen zugewiesen. In einer For-Scheife wird die Textdatei mit allen Werten überschrieben, sodass die fehlende Zeile durch eine andere ersetzt wird. Wenn die Schleife komplett durchlaufen ist werden die geöffneten Dateien durch die Methode „close()“ geschlossen. Das Ganze wird analog für die anderen Textdateien gemacht. Wenn all diese Schritte erfolgreich abgearbeitet sind, wird dem Nutzer eine Nachricht über das erfolgreiche Löschen seines Eintrags angezeigt.

4.5 Befehl: „copy“

Mit dem Befehl „copy“ wird auf die Funktion **„copy()“** in „module_direkt“ zugegriffen.
Mit dem Befehl „copy“ kann man mit dem Argument „—titel“ oder „-t“ nach einem gewissen Titel suchen, der vorher bereits im System erfasst wurde und mit einem dazugehörigen Username und Passwort gespeichert ist. Zu Beginn wird die Eingabe des Masterpassworts mit der Funktion **„masterpw_vergleich()“** erfordert. Ist diese Eingabe korrekt wird mit dem Ablauf fortgefahren. Anschließend wird mit der Funktion **„suche_titel()“** geprüft, ob die Eingabe des Titels mit einem bestehenden Titel übereinstimmt. Falls dies der Fall ist, wird das dazugehörige Passwort mit der Funktion **„dec()“** entschlüsselt und der entsprechende Username ausgegeben. Das Passwort wird anschließend in die Zwischenablage für 15 Sekunden mit der Funktion **„kopieren()“** kopiert.
Falls die Suche nach dem Passwort fehlgeschlagen ist, wird folgende Fehlermeldung ausgegeben: „Titel existiert nicht! Probieren Sie einen anderen Titel“.

4.6 Unbekannter Befehl

Für unbekannte Befehle ist die Funktion **„Info()“** aus „module_direkt“ zuständig. Diese Funktion besteht nur aus Printfunktionen, die dem Nutzer bei Eingabe eines unbekannten Befehls einer Übersicht gibt, welche Befehle existieren und was die Aufgaben der einzelnen Befehle ist.

5. Zusatzfunktion

In einer der gegebenen Aufgabestellungen wurde nach einer Erweiterung des Programms bzw. einer weiteren Funktion gefragt. Für diese zusätzlichen Anforderungen liegt der Schwerpunkt auf ein gut geschütztes und sicheres Masterpasswort. Als Beispiel wurde zum Vergleich einige andere Passwort-Sicherheitsabfragen angeschaut, wie beispielsweise Google, Facebook oder andere soziale Netzwerke. Wir haben uns dann für eine

Sicherheitsabfrage entschieden, die vom User beantwortet werden muss. Diese Frage wird auch vom User erstellt. Es wird eine „frage.txt“-Datei und eine „antwort.txt“-Datei erstellt. Mit der Funktion „createmasterpas“ kann nun ein neues Masterpassword erstellt werden und eine Sicherheitsfrage hinzufügt werden. Zur Änderung des Masterpassworts braucht man unter anderem das alte Passwort und muss zusätzlich die gegebene Sicherheitsfrage richtig beantworten. Nachdem der User diese Anforderungen erfüllt hat, kann das Masterpasswort geändert werden.
Eine weitere Zusatzfunktion, die für den Passwort-Manager erstellt wurde, ist auch eine sehr wichtige Funktion, die in jedem Passwort-Verwaltungsprogramm vorhanden sein sollte, um den User über seine Sicherheit Aufmerksam zu machen. Dabei wird die Stärke des Masterpasswords durch 1 bis 3 Sterne „*“ dargestellt. Einen Stern bekommt man, wenn das Passwort länger ist als sieben Zeichen. Einen weiteren, wenn mindestens ein Großbuchstabe enthalten ist und für drei Sterne muss zusätzlich mindestens eine Zahl vorkommen. Die Passwortbedingungen für ein sicheres Passwort werden dem Nutzer vor der Passworteingabe angezeigt. Nach der Eingabe des ausgewählten Passworts wird dem Nutzer seine Passwortstärke mitgeteilt. Sollte keine der Bedingungen erfüllt sein, wird dem Benutzer die Passwortstärke unsicher angezeigt. Diese Zusatzfunktion ist sehr sinnvoll, da dadurch der Nutzer eine gute Auskunft über die Sicherheit des Passwortes erhält und somit gut von einem starken und schwachen Passwort unterscheiden kann.

6. Feedback

6.1 Probleme/Herausforderungen

Generell muss ehrlicherweise gesagt werden, dass dieser Passwort-Manager so nicht wirklich realitätstauglich ist. Es wäre nicht empfehlenswert, wichtige Passwörter auf diese Weise zu speichern. Zwar ermöglicht das Masterpasswort nur dem richtigen Nutzer einen Einblick auf die Konsole, allerdings kann in unserem Fall sehr leicht auf die ungeschützten Textdateien zugegriffen werden, in denen die Passwörter gespeichert sind. Außerdem kann der Inhalt auch einfach gelöscht werden. Ein möglicher Lösungsansatz wäre, schreibgeschützte Dateien zu verwenden. Leider haben wir es in der Zeit nicht mehr geschafft, das Ganze in Python zu realisieren.
Dennoch sind die Passwörter in unserem Programm durch das Verschlüsseln/Encrypten ziemlich gut gesichert. Die Passwörter können zwar wieder entschlüsselt werden, doch dies kann nur mit der Eingabe des Masterpassworts erfolgen. Daher haben wir das Masterpasswort nicht nur encryptet, sondern gehasht. Dadurch besteht nicht die Möglichkeit das Masterpasswort nach dem Verschlüsseln wieder zu entschlüsseln. Es kann lediglich nur mit der Funktion „masterpw_vergleich“ geprüft werden, ob das eingegebene Masterpasswort richtig ist.
Die einzige Anforderung, die unserem Programm fehlt, ist die Masterpasswort-Abfrage. In unserem Fall wird das Passwort tatsächlich vor jeder einzelnen Aktion angefordert. Obwohl mehrfach versucht wurde, einen Timer zu installieren, gelang es uns letztendlich nicht, ihn in den Programmcode zu integrieren.

6.2 Fehlende Anforderung

Erst wurde versucht das mittels der „time.sleep()“ Methode und dem Modul „os“ wie bei der Funktion „kopieren()“ zu realisieren, was jedoch nicht zu unserem gewünschten Ergebnis geführt hat. Uns wurde schnell bewusst, dass ein weiteres Programm nebenbei laufen muss, da mit der „time.sleep()“ Methode das ganze Terminal einfriert und man in dieser Zeit nicht im Terminal arbeiten kann. Die Idee war es ein Kindprozess zu erstellen, der die Zeit ständig abfragt und nachschaut, ob die fünf Minuten um sind. Allerdings müssten hierzu dann in fast allen Funktionen, in denen das Passwort abgefragt wird, die Funktion der Passwortabfrage in der Zeit stillgelegt werden, was uns nicht gelungen ist.
Eine weitere Herausforderung dabei ist die Anforderung so zu realisieren, dass die ganzen bereits bestehenden und funktionierenden Funktionen/Anforderungen weiterhin funktionieren.
Auch müsste die Zeit der Passworteingabe irgendwo gespeichert werden, damit das Programm die Zeit abgleichen kann. Dazu hätte man die Zeit in einer Textdatei abspeichern können, die Zeit dann immer aus der Textdatei abzugleichen.
Bei der Recherche kam auch das „Threading“ oft als Lösungsansatz vor, was einen parallel-laufenden Code innerhalb desselben Codes ermöglicht. Allerdings gab es auch hierbei Probleme bei der Realisierung und dieselben Probleme wie bei der Idee mit dem Kindprozess.

6.3 Fazit

Zu Beginn der Implementierung des Passwortmanagers haben wir zunächst versucht alle unsere Ideen zu programmieren. Im Endeffekt haben wir jedoch gemerkt, dass wir auf jeden Fall zuerst ein gutes Konzept brauchen, nach dem wir besser arbeiten sollten. Diese Struktur hat uns geholfen das Programm als Ganzes zu betrachten und alle Anforderungen aufeinander abzustimmen. Nachdem wir uns alle erstmal an Python, womit wir zuvor noch nicht gearbeitet hatten, gewöhnt haben, wurde anfangs hauptsächlich recherchiert. Als wir ausreichend Informationen gesammelt hatten, wie gewisse Funktionen umgesetzt werden könnten, ging es an die tatsächliche Programmierung. Abschließend kann gesagt werden, dass wir durch das Projekt sehr viel dazu gelernt haben. Die Auseinandersetzung mit dem Thema hat uns auf jeden Fall gezeigt, dass ein gutes Konzept eine anfangs sehr komplex wirkende Aufgabe sehr erleichtern kann. Es hat uns sehr viel Spaß gemacht, da wir nun am Ende ein fertiges, funktionsfähiges Programm präsentieren können, das auch im Alltag durchaus sinnvoll gebraucht werden kann.

7. Quellen

1. https://docs.python.org/3/howto/argparse.html

2. https://www.delftstack.com/de/howto/python/python-find-string-in-file/

3. https://lanbugs.de/howtos/python-co/python-snippet-in-einer-datei-suchen-und-zeilennummern-zurueckgeben/

4. https://dbader.org/blog/python-commandline-tools-with-click

5. https://www.delftstack.com/de/howto/python/how-to-read-specific-lines-from-a-file-in-python/

6. https://www.delftstack.com/de/howto/python/python-get-number-of-lines-in-file/

7. https://www.delftstack.com/de/howto/python/data-in-table-format-python/

8. https://www.python-kurs.eu/forking.php

9. https://pypi.org/project/stdiomask/

10. https://www.it-swarm.com.de/de/python/wie-pruefe-ich-ob-eine-datei-leer-ist-oder-nicht/968141004/

11. https://www.base64encoder.io/python/

12. https://riptutorial.com/python/example/27070/encoding-and-decoding-base64

13. https://pypi.org/project/bcrypt/

14. https://stackoverflow.com/questions/59363298/argparse-expected-one-argument

15. https://stackoverflow.com/questions/33560802/pythonhow-os-fork-works/33561147

16. https://pypi.org/project/pyperclip/

17. https://chriswarrick.com/blog/2014/09/15/python-apps-the-right-way-entry_points-and-scripts/

18. https://www.geeksforgeeks.org/file-handling-python/

19. https://www.programiz.com/python-programming/time/sleep