Frankfurt University
of Applied Sciences
Frankfurt am Main
Betriebssysteme und Rechnernetze

Passwortmanager

– Werkstück A –

Von: Rania Hajjout (1360147)
Issam Boutachdat (1352294)
Luke Cefariello (1360053)

E-Mail: Rania.hajjout@stud.fra-uas.de
Issam.boutachdat@stud.fra-uas.de
Luke.cefariello@stud.fra-uas.de

Themensteller: Prof. Dr. Christian Baun

Vorgelegt am: 28. Juni 2021 in Frankfurt am Main

Heutzutage ist man bei vielen Plattformen angemeldet, wofür man meistens einen Benutzername und ein dazugehöriges Passwort benötigt. Jedoch ist hierfür das Problem, dass der menschliche Verstand oftmals das Passwort vergisst, sowie auch den eigenen Benutzernamen. Um diese Probleme zu vermeiden, gibt es den Passwortmanager, welcher Daten sicher und persistent speichert. Unter anderem soll ein Passwortmanager verschiedene Features anbieten, wie ein sicheres Passwort generieren zu lassen, um dieses in Zukunft verwenden zu können. Anhand dieser Gründe hat sich unsere Gruppe für das Projekt, Passwortmanager, entschieden, da dieses Thema realitätsnah ist und dieses wiederum unser Interesse geweckt hat.
Um den Passwortmanager zu verbessern, haben wir uns Zusatzfunktionen überlegt. Dazu gehört ein Timer, welcher die Aufgabe hat, den Nutzer zu erinnern sein Passwort zu ändern, wenn das gespeicherte Passwort veraltet ist, um die Sicherheit des Passwortes zu verstärken. Eine weitere Zusatzfunktion ist die Passwortsicherheitsstufe, welche nach dem Speichern oder Generieren eines Passworts die entsprechende Stärke hierfür bekannt gibt, um den Nutzer eventuell zu motivieren, das jeweilige Passwort zu ändern, falls es für schwach beurteilt wird.
Ebenfalls haben wir uns gefragt, wie man das Masterpasswort und die eingespeicherten Passwörter sicher und verschlüsselt speichern kann. Demzufolge haben wir dies als strittige Frage beurteilt und als große Herausforderung angenommen. Beim Programmieren des Passwortmanagers haben wir uns für die Programmiersprache Python entschieden, da wir grundlegendes Wissen aus der Sprache Java haben und eine abgewandelte Form dessen erlernen wollten. Zudem ist Python eine sehr angesehene Programmiersprache.
Im Folgenden werden Methoden kursiv und Variablen unterstrichen dargestellt.

In der Main werden Anfangs mehrere Variablen deklariert, auf die bei erster Nutzung eingegangen wird.
Zuerst werden die nötigen Dateien, mit Hilfe der Methode *datei_erstellen(),*erstellt. In dieser Methode wird die Datei mit den Nutzerdaten und die Datei mit der Passwort Prüfung erstellt, indem diese Dateien im Appending-Mode geöffnet werden. Falls diese Dateien schon existieren, passiert nix.
Daraufhin wird ein Objekt von der Klasse Nutzerdaten erstellt, und somit auch die dazugehörige Methode *__init__* aufgerufen. Bei der Methode werden die Attribute <u>title</u>, <u>code</u> und <u>username</u> deklariert.
Als nächstes wird die Methode *mpasswort_erstellen()* aufgerufen, in der ein Masterpasswort erstellt wird und die dazugehörigen Schritte durchgeführt werden.
Zum Anfang der Methode beginnt eine While-Schleife, welche erst beendet wird, wenn ein Masterpasswort vorhanden wird. Solange dies nicht geschieht, ist man in der Schleife „gefangen“. Ob das Masterpasswort vorhanden ist, wird mit der Methode *master_vorhanden()* geprüft.
In der *master_vorhanden()*-Methode wird überprüft, ob die Datei mit der Masterpasswortprüfung leer ist. Wenn in der Datei die Masterpasswortprüfung gespeichert wird, wird daraufhin ein Masterpasswort erstellt, also muss die Datei leer sein, falls es kein Masterpasswort gibt.

In der Methode *mpasswort_erstellen()* geht es weiter damit, dass zunächst darauf hingewiesen wird, dass ein Masterpasswort erstellt werden muss und daraufhin das gewünschte Masterpasswort eingegeben und bestätigt werden muss. Falls die Eingaben nicht übereinstimmen, wird dies auch ausgegeben, falls die Eingaben übereinstimmen, wird ein Crypter erstellt, dabei wird die Methode *crypter_erstellen()* aufgerufen und als Parameter die Eingabe des gewünschten Masterpasswortes übergeben.

Der Parameter der *crypter_erstellen()*-Methode ist das Masterpasswort. Zu Beginn wird mit der hashlib die Methode *sha256()* aufgerufen und das Masterpasswort in Bytes übergeben. Somit ist das Masterpasswort als „hash" in der Variable pw_hash gespeichert. Mit dieser Variablen rufen wir nun die Methode *hexdigest()* auf, woraufhin die ersten 32 Zeichen in der Variable pw_hexi gespeichert werden. Dies wird gemacht, weil *Fernet()* nur url-32 annimmt. Als nächstes wird der Parameter pw_hexi der Methode b64encode() übergeben, um anschließend dies Fernet() zu übergeben. Dadurch wird ein Crypter erstellt, welcher im Voraus global deklariert wurde.

Als nächstes wird in der Methode *mpasswort_erstellen()* die Methode *speicher_verschluesselte_nummer()* aufgerufen. Hierbei wird in der Masterpasswortprüfung-Datei die Zahlenreihe 12345 als String gespeichert. Dies wird gemacht, um im späteren Verlauf des Codes zu überprüfen, ob das Masterpasswort korrekt ist, indem die Zahlenreihe mit dem Crypter des Masterpasswortes verschlüsselt gespeichert wird. Beim Entschlüsseln muss mit dem aktuellen Masterpasswort die Zahlenreihe wieder „12345" ergeben. Falls dies der Fall ist, stimmt das Masterpasswort überein.

In der Main geht es weiter, indem das Masterpasswort eingegeben werden muss, wofür man 3 Versuche hat. Falls alle 3 Versuche falsch sind, wird das Programm beendet. Bei einer richtigen Eingabe wird auf die Eingabe eines Befehls erwartet. Der eingegebene Befehl wird dann ausgeführt. Sollte der Befehl nicht existieren, passiert nix und der User kann weiterhin Befehle eingeben. Nachdem ein Befehl ausgeführt wurde, muss nicht nochmal das Masterpasswort eingegeben werden, sondern kann direkt der nächste Befehl eingegeben werden. Das Masterpasswort muss jedoch nach drei Minuten wieder eingegeben werden, sobald ein Befehl beendet wird. Der Befehl **exit** beendet das Programm. Dies wird gemacht, indem zu Beginn des Ablaufs eine While-Schleife steht, welche mit der *master_vorhanden()*-Methode überprüft, ob ein Masterpasswort vorhanden ist. Zusätzlich wird geprüft, ob der Befehl nicht exit ist. Falls diese Bedingungen zutreffen, wird das Masterpasswort verlangt, nach jeder Eingabe die Versuche erhöht und ein Crypter erstellt, indem die Methode *crypter_erstellen()* aufgerufen wird. Falls das Passwort übereinstimmt, welches mit der Methode *passwort_pruefung()* geprüft wird, werden die Versuche auf 0 gesetzt, die aktuelle Zeit und der „Ablaufzeitpunkt" des Masterpasswortes gespeichert, der Befehl zurückgesetzt und der User drauf hingewiesen, dass das Masterpasswort akzeptiert wurde.

In der *passwort_pruefung()* Methode wird als Parameter das zu prüfende Masterpasswort übergeben. Als nächstes wird der Crypter, mithilfe der Methode *crypter_erstellen()* geändert. Daraufhin wird die

Datei, mit der Masterpasswortprüfung(„12345“) in einen Array gespeichert und der erste Wert, welcher „12345“ verschlüsselt enthält, an die Methode *entschluesseln()* übergibt.
In der *entschluesseln()*-Methode wird versucht den übergebenen verschlüsselten Parameter zu entschlüsseln, indem mit dem Crypter die Methode *decrypt()* aufgerufen wird und die gegebenen Bytes übergeben werden. Diese werden dann mithilfe der Methode *decode()* zu einem String umgewandelt und daraufhin zurückgegeben. Falls dies eine Fehlermeldung ergeben sollte, wird der User darauf hingewiesen, dass das eingegebene Masterpasswort fehlerhaft ist.
Der Methode *verschluesseln()* hingegen sollte ein String übergeben werden. Dieser String wird dann mit der Methode *encode()* in Bytes umgewandelt und mithilfe der Methode *encrypt()*, welchem die Bytes übergeben werden, verschlüsselt. Diese verschlüsselten Bytes werden dann zurückgegeben.
Als nächstes kommt eine While-Schleife, welche prüft, ob der aktuelle Befehl nicht „abgelaufen“ und nicht „exit“ lautet. Falls der Befehl jedoch „abgelaufen“ oder „exit“ ist, wird wieder der Prozess ab dem Eingeben des Masterpasswortes wiederholt. Der Befehl wird zu „abgelaufen“, wenn die aktuelle Zeit, die Zeit vom „Ablaufzeitpunkt“ übertrifft. Falls die Befehle nicht „abgelaufen“ oder „exit“ sind und das Masterpasswort auch nicht „abgelaufen“ ist, wird auf die Eingabe eines Befehls gewartet.
Die Befehle lauten **generate**, **store**, **delete**, **modify**, **list**, **search**, **change**, **check** und **exit.** Die Eingabe des Nutzers wird als String gespeichert und in mehreren If-Abfragen abgefragt. Sollte eine davon zutreffen, wird der jeweilige Befehl ausgeführt. Falls keiner der Befehle zutrifft, kann der User erneut einen Befehl eingeben.

Für den Befehl **generate** müssen verschiedene Module importiert werden, um im Code mit den Schlüsselwörtern arbeiten zu können. Diese wären „import random“, „import secrets“ und „import string“. Im Folgenden kann der Nutzer aus verschiedenen Optionen wählen und sein Passwort generieren lassen.
Zunächst wird der Nutzer gefragt, welche Länge das Passwort haben soll. Die Eingabe des Nutzers wird in einer Variablen gespeichert, welche im weiteren Verlauf genutzt wird.
Hierzu werden Anfragen gestellt, ob im Passwort Zahlen, Kleinbuchstaben, Großbuchstaben und Sonderzeichen enthalten sein sollen. Für die jeweiligen Anfragen werden eigene Variablen deklariert, in denen die eigenen Zeichenarten gespeichert sind. Für Zahlen speichert man „string.digits“, für Großbuchstaben „string.ascii_uppercase“ und Kleinbuchstaben „string.ascii_lowercase“ aus der ASCII-Tabelle und für Sonderzeichen eigene gewählte Sonderzeichen in einem String.
Um das Passwort mit den Anforderungen des Nutzers zusammenstellen zu können, deklariert man erstmal eine Variable „kombiniert“ mit einem leeren String, um nachfolgend die jeweiligen Zeichen hinzufügen zu können.
Dementsprechend gibt es eine While-Schleife mit verschiedenen If-Abfragen. Die While-Schleife läuft so lang die gewünschte Länge des Passwortes größer als die Länge des generierten Passwortes ist. Bei der Wahl der Zeichen, wird mit der Methode secrets.*choice*(Variabelname) jeweils zufällig ein Zei-

chen aus den bestehenden Zeichen-Variablen gezogen und der Variablen kombiniert hinzugefügt. Nachfolgend werden die Zeichen, die in der Variablen kombiniert gespeichert sind, durchgemischt und in eine neue Variable gespeichert, sodass jedes Zeichen auch einen anderen Index bekommt.
Zudem soll der Passwortgenerator so gebaut sein, dass er unproblematisch ist, sodass man keine Möglichkeit hat ein Passwort generieren zu lassen, dass kleiner sein soll als die Anzahl der Anforderungen. Deshalb wird die Anzahl der Anforderungen mit der Passwortlänge verglichen.
Letzten Endes wird die Methode *zwischenablage_speichern(pw)* aufgerufen, um das generierte Passwort in die Zwischenablage zu speichern, und die Passwortstärke des generierten Passworts mit der Methode *passwortsicherheit(passwort)* ausgegeben.
Die Methode *passwortsicherheit(passwort)* enthält die Variable staerke, in der durch If-Abfragen das Zeichen „*“ hinzugefügt wird. Besteht das Passwort aus Groß- oder Kleinbuchstaben, einem Sonderzeichen oder Zahlen, wird je Abfrage ein „*“ hinzugefügt.

Unter Anderem soll ein Passwortmanager die Funktion haben, das angeforderte Passwort in die Zwischenablage zu speichern. In diesem Fall wird das Passwort für 30 Sekunden in der Zwischenablage gespeichert. Hierfür muss man die Module „import subprocess”, “import time” und “import threading“ importieren.
Daraufhin wird die Methode *zwischenablage_speichern(txt)* erstellt, in der man mit “echo“ und “| clip“ arbeitet. Mit “echo“ wird der Text in die „Console“ geschrieben. Das Zeichen “|“ sorgt dafür, dass der Output von dem „command“ links in den „command“ rechts als Input übergeben wird. Mit “clip“ wird der Text in die Zwischenablage gespeichert wird. Dieser wird in einer Variable ablage gespeichert. Um das Passwort schlussendlich in die Zwischenablage zu speichern, verwendet man die Methode *check_call(),* die durch subprocess aufgerufen wird. Dieser Methode wird ablage übergeben. Außerdem wird danach ein Objekt erstellt, dass in einer Variablen gespeichert ist. Dieses Objekt wird dann mit der Methode *start()* verbunden, welche die Aufgabe hat alle Methoden aus der Klasse Thread1 zu starten.
Darüber hinaus hat man hier auch eine Klasse „Thread1“ erstellt, die von Thread erbt. Diese Klasse enthält die Methode *__innit__(self),* welche für die Verbung des Konstruktors zuständig ist. Außerdem enthält die Klasse die Methode *run(self),* die für das Speichern des Passworts für 30 Sekunden zuständig ist. Dies wurde mit dem Code time.sleep(30) ermöglicht. Nachfolgend wird dann die Methode *zwischenablage_loeschen()* aufgerufen, die erst nach 30 Sekunden eingeschaltet wird. Die Methode sorgt dafür, dass nach 30 Sekunden eine Information an den Nutzer gegeben wird, dass das Passwort aus der Zwischenablage gelöscht worden ist.

Um Farben bei Windows einsetzen zu können muss man zunächst über den Registrierungseditor den Ordner „HKEY_CURRENT_USER“ und den Unterordner „Console“ öffnen. Dann muss ein neuer „DWord-Wert“ erstellt werden, den man „VirtualTerminalLevel“ nennt und den Binärwert auf „1“

umändern. Nachfolgend soll über die CMD „colorama“ installiert werden, sodass das Ganze später funktioniert.
Im Python-Code muss das Modul „from colorama import Fore“ importiert werden. Schlussendlich muss man alle Wörter beziehungsweise Zeichen, die in der Shell farbig ausgegeben werden sollen, mit den Schlüsselworten wie zum Beispiel „Fore.LIGHTGREEN_EX + ‚Titel‘ + Fore.RESET“ ergänzen.

Der Befehl **store** ist dafür da, ein Objekt zu speichern, sodass man Daten eingibt, die man speichern möchte. Hierbei soll man erstmals die Inputs erstellen, indem man den Titel, Username und das Passwort eingibt. Dabei muss logisch geprüft werden, ob bereits derselbe Titel gespeichert wurde, was mit der Methode *exist(titel)* geprüft wird. Ist derselbe Titel bereits vorhanden, wird die Eingabe nicht gespeichert und der Nutzer bekommt die Information, dass der entsprechende Titel bereits vorhanden sei. Dementsprechend ordnet man die Eingaben des Nutzers, dem Objekt „nutzerdaten1“ zu und die Passwortstärke *passwortsicherheit(passwort)* des eingegeben Passworts wird ausgegeben. Außerdem wird die Methode *daten_hinzufuegen(self)* aufgerufen und mit dem Objekt „nutzerdaten1“ verknüpft, um Daten in eine Textdatei einzuspeichern.
Bei der Methode *pruefe_einagbe(self)*, erstellt man zunächst eine Variable, in der alle zuvor eingegebenen Daten, wie der Titel, der Benutzername, und das Passwort gespeichert sind. Weiterhin wird eine Variable deklariert, in der man die Daten mit einem *split(„ “)* zerteilt, um im Folgenden zu prüfen, ob der Nutzer bei der Eingabe Leerzeichen verwendet hat. Sollte dies der Fall sein, wird „False“ als Rückgabewert zurückgegeben.
Die Methode *exist(titel)* kontrolliert , ob ein Titel bereits vorhanden ist oder nicht. Am Anfang wird die Textdatei im Read-Bytes-Mode und mit der *readlines()*-Methode geöffnet, die die Zeilen in einer Variablen speichert. Danach werden die Daten mithilfe der Methode *entschluesseln(Variablelname)* entschlüsselt und nach dem Leerzeichen zerteilt. Somit besteht die Möglichkeit den ersten Eintrag pro Zeile herauszulesen. Im Folgenden nutzt man die Methode *lower()*, um alle Titel vollständig in Kleinbuchstaben umzuändern. Letzten Endes prüft man die Titel nach Gleichheit, sodass „True“ zurückgegeben wird, falls ein Titel bereits vorhanden ist.
In der Methode daten*_hinzufuegen(self)* muss eine Variable erstellt werden, in der das aktuelle Datum gespeichert wird. Unter anderem muss man die Methode *pruefe_eingabe()* in einer If-Abfrage aufrufen. Sobald die If-Abfrage „True“ als Rückgabewert liefert, öffnet sich eine Textdatei, welche die zugordneten Daten von dem Objekt „nutzerdaten1“ über den Appending-Bytes-Mode, verschlüsselt in Bytes, speichert.

Der Befehl **Delete** ist dafür da, einzelne Nutzerdateien zu löschen. Dies erfolgt über die Methode *nutzerdatei_loeschen()*, die aufgerufen wird, sobald der Nutzer „Delete“ als Befehl eingibt. Grundsätzlich wird der Inhalt der Datei „Testen.txt“ vollständig gelöscht und anschließend vereinzelt wieder reingeschrieben, bis auf die Nutzerdatei, die gelöscht werden soll.

Zunächst wird der Nutzer durch einen Input gefragt, welchen Titel er löschen wolle. Dieser Titel wird in title_delete gespeichert. Daraufhin folgt eine If-Abfrage, die überprüft, ob der angegebene Titel in der Datei „Testen.txt“ vorhanden ist. Dafür wird die Methode *exist()* verwendet und der eingegebene Titel übergeben. Ist der Titel in „Testen.txt“ zu finden, geht das Programm in den If-Zweig rein. Ist der Rückgabewert „False“, wird ausgegeben, dass der Titel nicht vorhanden sei.
Nun folgt ein weiterer Input, in dem zur Bestätigung ein weiteres Mal nachgefragt wird, ob man die Nutzerdatei mit dem Titel unwiderruflich löschen solle. Die Eingabe wird in der Variablen bestaetigung gespeichert.
In der nächsten Abfrage wird überprüft, ob die Variable dem String „Bestaetigen“ entspricht. Wenn der Boolean-Wert „True“ zurückgibt, geht er wieder in den If-Zweig. Falls der Boolean-Wert „False“ zurückgibt, wird ausgegeben, dass die Löschung nicht bestätigt wurde. Nun wird die Datei „testen.txt“ in Bytes gelesen. Alle vorhandenen Zeilen werden in dem Array lines gespeichert, woraufhin die Datei wieder geschlossen wird. Anschließend wird der Inhalt der gesamten Datei gelöscht, indem man sie im Writing-Mode öffnet und die Methode *close()* verwendet. Im nächsten Schritt gelangt man in die For-Schleife, in der man aus jedem Index, also „bit_line“, des Arrays lines die Daten entschlüsselt. Dies passiert mithilfe der Methode *entschluesseln()*, indem die Bytes wieder in Strings umgewandelt werden . Der übergebene Parameter ist bit_line. Nun wird im Array userdaten der Inhalt von bit_line gespeichert, der durch Leerzeichen getrennt wird. Im Folgenden untersucht man, ob der Titel, den man löschen möchte, nicht der String im Index Null von userdaten ist. Wenn die If-Abfrage den Wert „True“ zurückgibt, wird die Datei im Appending-Bytes-Mode geöffent. Nun wird die Datei wieder mit der Zeile beschrieben und ein Zeilenumbruch eingefügt, da der gesuchte Titel nicht gefunden wurde. Dies wird für jeden Index in „lines“ fortgeführt.

Der Befehl **Modify** ist dafür da, das Passwort einzelner Nutzerdateien zu ändern. Hierbei wird die Methode *nutzerdatei_aendern()* verwendet. Das Prinzip ist ähnlich wie bei der Methode *nutzerdatei_loeschen()*.
Zunächst wird nach dem Titel gefragt, wo eine Änderung gewünscht ist. Dieser Input wird in titel gespeichert. Danach überprüft man mithilfe der Methode *exist()*, ob der Titel überhaupt vorhanden ist. Falls dies nicht der Fall ist, wird ausgegeben, dass der Titel nicht vorhanden sei. Wird aber der Boolean-Wert „True“ zurückgegeben, so soll man den Benutzernamen unter username und das neue Passwort unter code eingeben. Das Passwort wird in einem String gespeichert und durch die Methode *getpass()* versteckt.
Im Folgenden geht man weiter wie bei dem Befehl **Delete**, sodass man den Inhalt zunächst löscht und dann wieder hinzufügt, wenn der Titel nicht in der jeweiligen Zeile zu finden ist.
Jedoch wird hierbei ein Else-Zweig hinzugefügt, wenn der Titel in userdaten vom Index Null ist. Hierbei wird die Datei im Appending-Bytes-Mode geöffnet und die Zeile wird verschlüsselt mit dem Ti-

tel, dem Username, dem neuen Passwort und dem Erstellungsdatum überschrieben. Danach wird ein Zeilenumbruch eingefügt.

Der Befehl **List** ist dafür da die ganze Liste der Nutzerdaten als Tabelle auszugeben. Dieser Befehl erfolgt über die Methode *ganze_Liste_ausgeben()*, die von dem Objekt nutzerdaten1 aufgerufen wird.
In dieser Methode wird die Datei „Testen.txt“ im Reading-Bytes-Mode geöffnet und alle Zeilen im Array lines gespeichert. Daraufhin wird die Datei wieder geschlossen.
Nun werden die Tabellenüberschriften erstellt. Dies erfolgt über eine Ausgabe mit der Methode *abstand()*, der ein Wort übergeben wird. Die Methode überprüft die Länge des übergebenen Wortes und übergibt dann das Wort und je nach Länge des Wortes null bis vier Tabs. Die Ausgabe beinhaltet einmal „Titel“ in gelb, „Username“ in grün, „Passwort“ in rot und „Erstellungsdatum“ in Magenta. Die Farben werden nach jedem Einsatz wieder zurückgesetzt. Daraufhin wird in der nächsten Zeile ein langer Balken eingefügt, um die Tabellenüberschrift zu verstärken.
Im nächsten Schritt wird in der For-Schleife jede Zeile der Datei einzeln ausgegeben, indem man sie zunächst entschlüsselt, die einzelnen Daten im Array userdaten speichert und dann mithilfe der Methode *abtsand()* die Daten in eine print-Anweisung setzt. Hierbei greift man auf die jeweiligen Indexe zu, um den Titel, den Username, das Passwort und das Erstellungsdatum auszugeben. Das Passwort wird mit der Methode *verdecken()* hinter Sternen versteckt, deren Anzahl mit der Länge des Passwortes übereinstimmen. Diese werden in der Variablen geheim_pw gespeichert und als String weitergegeben. In der passenden Farbe zur Tabellenüberschrift werden die Angaben ausgegeben.
Das Ergebnis dieser Methode ist eine geordnete Tabelle nach „Titel“, „Username“, „Passwort“ und „Erstellungsdatum“.

Der Befehl **Search** ist dafür da, einzelne Nutzerdateien zu finden und ausgeben zu lassen. Mithilfe der Methode *nutzerdatei_finden()* wird dieser Befehl realisiert.
Zunächst wird ein Titel eingelesen, der in titel gespeichert wird. Zusätzlich wird die Datei im Reading-Bytes-Mode geöffnet. Daraufhin überprüft man, ob der angegebene Titel in der Datei zu finden ist. Wenn dies nicht der Fall ist, wird ausgegeben, dass der Titel nicht vorhanden sei. Wird der Titel gefunden, geht es im If-Zweig weiter, wo alle Zeilen im Array lines gespeichert werden. Danach wird die Datei geschlossen und die For-Schleife beginnt, wie zuvor bei den anderen Befehlen. Für jeden Index wird der Inhalt entschlüsselt und im Array userdaten gespeichert. Im Folgenden wird überprüft, ob der gesuchte Titel gleich dem ersten Index von userdaten ist. Ist der Boolean-Wert „True“, so wird die Nutzerdatei ausgegeben. Es wird der Titel in gelb, der Username in grün und das versteckte Passwort in rot ausgegeben. Das Passwort wurde wieder durch die Methode *verdecken()* versteckt. Zudem wird das Passwort mit Hilfe der Methode *zwischenablage_speichern()* in der Zwischenablage gespeichert.

In dem Befehl **Change** wird das Masterpasswort geändert und somit auch alle Daten neu verschlüsselt. Es wird die Methode *masterpasswort_aendern()* aufgerufen. Zunächst wird das alte Passwort gefordert, um die Identität zu bestätigen. Dies wird mit der Methode *passwort_pruefung()* überprüft. Falls das Passwort Falsch ist, wird der Nutzer darauf hingewiesen, wenn es jedoch richtig ist, soll der Nutzer sein neues Passwort zweimal eingeben. Sollte die zweimalige Eingabe des neuen Passwortes nicht übereinstimmen, so wird darauf hingewiesen. Stimmen die Eingaben überein, sollen die Dateien neu mit der Methode *datei_neu_verschluesseln()* verschlüsselt werden, dieser Methode wird das alte Masterpasswort, das neue Masterpasswort und der Name der Datei mit den Nutzerdaten übergeben. Beim zweiten Aufruf der Methode wird, an Stelle der Datei mit den Nutzerdaten, die Datei mit der Passwortüberprüfung übergeben.
Beim ersten Aufruf ist der Crypter zwar noch über das alte Masterpasswort erstellt worden, jedoch wurde er beim zweiten Aufruf durch das neue Masterpasswort erstellt, welches aber nicht gewünscht ist. Aufgrund dessen wird anfangs der Crypter geändert. Hierfür wird die Methode *crypter_erstellen()* aufgerufen und das alte Masterpasswort übergeben. Die Daten der aktuellen Datei werden als Bytes in einem Array lines mithilfe der Methode *readlines()* gespeichert. In einer For-Schleife wird jeder Wert von lines mit der Methode *entschluesseln()* entschlüsselt und in einem Array daten gesichert. Als nächstes wird der Crypter geändert, indem die Methode *crypter_erstellen()* aufgerufen und das neue Masterpasswort übergeben wird. Nun wird die alte Datei gelöscht, indem man die Methode *loesche_datei()* aufruft und den jeweiligen Dateinamen übergibt. Zuletzt geht eine For-Schleife das Array daten durch und speichert die jeweilige datei, in diesem Beispiel den Titel, den Nutzernamen, das Passwort und das Erstellungsdatum, verschlüsselt in die zugehörige Datei ab. Verschlüsselt wird mit der Methode *verschluesseln()*, übergeben wird die Variable datei. Nach jeder Speicherung wird ein Zeilenumbruch gemacht.

Bei dem Befehl **Check** wird grundlegend geprüft, ob die gespeicherten Passwörter „abgelaufen“ sind. „Abgelaufen“ sind Passwörter, welche 30 Tage nicht verändert wurden. Dies geschieht mithilfe der Methode *pruefe_abgelaufene_passwoerter()*.
Man beginnt damit alle Daten der Datei „Testen.txt“ in einem Array in Bytes zu speichern und einen Boolean zu deklarieren, welcher prüfen soll, ob es mindestens ein Passwort gibt, welches abgelaufen ist. Daraufhin geht man mit der For-Schleife jeden Array-Wert, in unserem Fall jede Zeile, durch. Jede Zeile wird entschlüsselt und zusätzlich werden die Nutzerdaten zerteilt in Titel, Nutzername, Passwort und Erstellungsdatum.
Daraufhin wird geprüft, ob das Erstellungsdatum des aktuellen Titels schon abgelaufen ist, indem das Erstellungsdatum von einem String zu einem „datetime“ Datentypen konvertiert wird. Hierbei wird die Methode *strptime()* aufgerufen, welche aus der datetime Bibliothek stammt, die man importieren muss. Das Erstellungsdatum wird dann um 30 Tage erhöht und mit dem aktuellen Datum verglichen.

Falls das Passwort abgelaufen ist, wird der Titel mit dem Nutzernamen und einer Aufforderung zum Ändern des aktuellen Passwortes ausgegeben. Dazu wird der oben genannte Boolean „True“.
Nachdem die Schleife durchlaufen ist, wird überprüft, ob der Boolean noch „False“ ist. Sollte dies zutreffen, wird darauf hingewiesen, dass alle Passwörter noch gültig sind.

Nach einer langen Auseinandersetzung mit der Programmiersprache Python konnten wir unser Projekt, den Passwortmanager, realisieren. Es wurden uns viele Möglichkeiten bereitgestellt, womit wir vielseitig arbeiten konnten. Grundsätzlich hat uns unser Vorwissen der Sprache Java sehr weiter geholfen.
Es gibt eine Vielzahl an Modulen, die die Lösung von Problemen deutlich vereinfacht haben. So konnten wir, wie anfangs schon beschrieben, die Passwörter verschlüsselt abspeichern. Dies gilt für das Masterpasswort, als auch für die eingegebenen oder generierten Passwörter.
Das zeitliche Problem hat leider dazu geführt, dass wir uns nicht darauf fokussieren konnten, die Dateien ausreichend zu schützen, sodass man von außen darauf zugreifen kann. Jedoch konnten wir die Passwörter so verschlüsseln, dass man sie nicht erkennen kann, wenn man die Datei öffnen würde.
Es war eine Herausforderung den Passwortmanager so zu programmieren, dass er benutzerfreundlich und zugleich sicher ist.
Schlussendlich haben wir es geschafft einen Passwortmanager zu kreieren, der alle gegebenen Anforderungen erfüllt und zusätzlich zwei weitere Funktionen beinhaltet, die das ganze Programm abrunden.