Inhaltsverzeichnis

Menü

Im Menü werden verschiedene Optionen zu Verfügung gestellt. Je nach dem Interesse wird dann die jeweilige Option ausgewählt.

```
----------PASSWORTMANAGER---------- \n\
Bitte wähle eine Option: \n\
    1. Füge ein neuen Eintrag hinzu \n\
    2. Siehe alle Einträge ein \n\
    3. Lösche einen alten Eintrag \n\
    4. Ändere Masterpasswort \n\
    5. Prüfe Passwörter nach Duplizität \n\
    6. Prüfe Passwörter, die kürzer als 6 Zeichen sind \n\
    7. Lösche alle Einträge\n\
    8. Suche nach einem Eintrag, um das Passwort zu kopieren
    9. Beenden
```

Der Code `\n\` sorgt dafür, dass der Text in einer neuen Zeile ausgegeben wird.

Zuerst wurde im Menü eine while Schleife implementiert. Solang der User die Zahl 9 nicht eingibt wird er immer wieder auf das Hauptmenü zurückgeleitet und muss eine Option wählen.
Menü Punkt 1 der Nutzer soll ein Passwort, Benutzername und Titel eingeben. Dazu hat er noch die Möglichkeit sich ein Passwort generieren zu lassen.

Menü Punkt 2 gibt alle Einträge aus. Genaue Funktionsweise -> Funktionsweise SQL S.5

Menü Punkt 3 ermöglicht es einen Eintrag zu löschen. Der Benutzername und Titel muss eingegeben werden damit das Programm den gewollten Eintrag löschen kann und es zu keiner Überlappung kommt. Durch die Notwendigkeit beide einzugeben wird dies verhindert. Benutzer soll nochmal bestätigen, dass sein Eintrag unwiderruflich gelöscht werden soll.

Menü Punkt 4 Masterpasswort -> S.5

Menü Punkt 5 dem Benutzer werden alle doppelten Passwörter ausgegeben, die das Programm in der Datenbank findet.

Menü Punkt 6 Passwörter werden auf ihre Zeichenlänge überprüft. Alle Passwörter, die weniger als 6 Zeichen haben, werden in Folge dessen ausgegeben.

Menü Punkt 7 hat die gleiche Funktion, wie Menü Punkt 3. Hier bezieht es sich aber auf alle Einträge.

Menü 8

Um die Passwörter in die Zwischenablage zu speichern, wurde Pyperpclip Module heruntergeladen. Die Installation wurde durch pip command im Terminal vollzogen. Dazu öffnen wir erstmal das Terminal, gehen dann in Python rein, drücken auf View -> Tool Windows und wählen Terminal aus. Mit dem jetzt hervorgekommenen Terminal Fenster wird der Befehl pip install pyperclip reingeschrieben. Dadurch fängt die Installation an und ist in wenigen Sekunden abgeschlossen. Nachdem angezeigt wird, dass Pyperclip erfolgreich geladen hat importieren wir Pyperclip. Im Terminal hat das Kopieren in die Zwischenablage funktioniert, jedoch nicht in der Kommandozeile, weshalb wir Pyperclip als Kommentar markiert haben. Zum Schluss rufen wir die Funktion copy auf (). Damit das Passwort auf dem Terminal nicht ausgegeben wird haben wir Funktion getpass verwendet.

Um für eine zeitliche Begrenzung in der Zwischenablage zu sorgen haben wir import time benutzt. Des Weiteren wurde eine while Schleife hinzugefügt, welche die Bedingung hat solange bis die Zeit auf null ist runter zu zählen.

```
1
2  import time
3  sec=60
4  while sec != 0:
5      sec = sec-1
6      time.sleep(1)
7      print(sec)
```

Menü Punkt 9 Programm wird durch exit beendet

SQL

Wir haben uns für eine SQL-Datenbank entschieden, da es viele Möglichkeiten bietet mit den eingespeicherten Tabellen zu arbeiten. Man kann einfach Tabellen mit verschiedenen Spalten hinzufügen, sie durchsuchen und sie wieder löschen. Man kann auch viele Zeilen, in unserem Fall Einträge, abspeichern und diese auch ohne großen Aufwand durchsuchen, sortieren nach bestimmten Eigenschaften oder sie wieder löschen. Aus diesen Gründen haben wir uns für eine SQL-Datenbank entschieden und erklären die Einzelheiten unserer Datenbank.

Alle Einträge im Passwort-Manager müssen persistent gespeichert werden können mit ihrem Titel, Benutzernamen und Passwort. Dazu haben wir eine SQL-Datenbank erstellt und sie nach Titel, Benutzername und Passwort eingeteilt.
Die Datenbankdatei heißt eintrag.db. Die Tabelle in der Datei heißt „pws“, was eine Abkürzung des Worts Passwörter darstellt. Unsere erste Spalte heißt Titel und damit sind die Namen der Internetseiten gemeint, wo man sich einloggen will. Man kann selbstständig den Namen eingeben z.B. „Snapchat, Instagram, GMX“. Diesen Eintrag kann man mit dem gleichen Namen mehrfach einspeichern, da es sein könnte, dass nur ein Benutzer mehrere Accounts besitzt.
Unsere nächste Spalte heißt Benutzernamen und hier haben wir einen primär Schlüssel vergeben, da ein Nickname oder eine Email-Adresse nur einmalig vorkommen darf. Deswegen sind hier keine mehrfachen Einträge möglich.
Unsere letzte Spalte heißt „pw“, was ebenfalls eine Abkürzung von Passwörter ist. Hier kann der Benutzer selber entscheiden, ob er sich ein neues Passwort generieren lassen möchte oder ein Selbstgewähltes einspeichern möchte.

Funktionsweise

Um eine Verbindung zu der SQL-Datenbank zu schaffen, muss man ein Objekt erstellen, dass diese Verbindung herstellt. Dazu haben wir das Objekt „Connection“ erstellt. Zuerst wird es mit der zweiten Pythondatei verbunden.

main.py

```
connection = db.connect()
```

In der zweiten Pythondatei befindet sich die Funktion, die „connect“ heißt.

db.py

```
def connect():
    return sqlite3.connect("eintrag.db")
```

Hier wird die SQL-Datenbank-Datei erstellt oder mit einer vorhandenen verbunden.

In der zweiten Pythondatei „db.py“ sind alle Befehle und Funktionen drin, die SQL betreffen. Die Befehle an SQL sind nach dem EVA-Prinzip getrennt gegliedert von den Funktionen, um eine bessere Übersicht zu erhalten. Die großgeschriebenen Variablen

```
INSERT_PW = "INSERT INTO pws (titel, benutzername, pw) VALUES (?, ?, ?);"
```

enthalten ein SQL-Befehl in sich.
Die Variable enthält einen SQL-Befehl. Dieser sagt, dass ein neuer Eintrag hinzugefügt werden soll. Da der Titel, Benutzername und das Passwort vom Benutzer frei eingegeben werden sollen, stehen bei „Values“ drei Fragezeichen. Das ist bereits so festgelegt, dass bei Fragezeichen, die Werte erst vergeben werden.

```
def add_pw(connection, titel, benutzername, pw,):
    with connection:
        connection.execute(INSERT_PW, (titel, benutzername, pw))
```

Die dazugehörige Funktion in Python sieht so aus. Für die beste Trennbarkeit und Übersicht gibt es für jeden Befehl eine Funktion. Um sie ausführen zu können sind Titel, Benutzername und das Passwort erforderlich. Diese werden vom Benutzer eingegeben und das Programm kann es zuordnen und hinzufügen. „Connection“ ist bei jedem SQL-Befehl erforderlich, um es auszuführen, weil es die Verbindung herstellt zwischen den beiden Pythondateien.

Das Hinzufügen eines neuen Eintrags erfolgt ohne einen return-Wert. Hier sind zwei weitere Beispiele aus dem Programm.

```
def get_all_pw(connection):
    with connection:
        return connection.execute(GET_ALL_PW).fetchall()
```

```
GET_ALL_PW = """
SELECT titel, benutzername
FROM pws;"""
```

Diese Funktion gibt alle Einträge aus der Tabelle wieder. Da alle Einträge mit dem Titel und Benutzernamen ausgegeben werden sollen, sind hier keine Fragezeichen. Der return-Wert gibt somit die Daten wieder und ist hier erforderlich. Zum Schluss steht noch .fetchall(). Das bedeutet es werden mehrere Einträge wiedergegeben. Im Umkehrschluss, wenn man weiß, dass man nur einen Eintrag bei einem bestimmten Aufruf sehen wird benutzt man .fetchone wie im unteren Beispiel.

```
def get_benutzerkonto(connection, titel, benutzername):
    with connection:
        return connection.execute(GET_BENUTZERKONTO, (titel, benutzername,)).fetchone()
```

Hier soll nur ein bestimmtes Benutzerkonto aufgerufen werden. Da es nur einmal vorkommen kann, wird hier .fetchone verwendet.

Zufälliger Passwortgenerator:

Wir haben einen Python-Code erstellt, wo der Benutzer ein je nach Wunsch zugfällig erstelltes Passwort ausgegeben bekommt.
Der Benutzer bekommt Optionen vorgegeben, die fragenartig gestellt werden. Er bekommt die Möglichkeit sich aussuchen, welche Bestandteile sein Passwort bekommen soll, er kann aussuchen, wie lang sein Passwort sein soll, ob

Großbuchstaben, Kleinbuchstaben, Sonderzeichen oder Zahlen in seinem zufällig generierten Passwort haben möchte.

Als erstes wird das „Secrets-Modul“ importiert:

```
import secrets
```

Das „Secrets-Modul“ wird zum Generieren von Zufallszeichen, wie z.B. Passwörter verwendet. Dieses Modul ist dafür verantwortlich, den Zufall aus verschieden Zeichen in verschiedenen Längen zu generieren.

Als nächstes müssen wir das Passwort als eine Variable initialisieren:

```
pw = ""
```

Wir geben dem Passwort einen beliebigen Variablennamen, wie z.B. „pw“.

Und beim initialisieren lassen wir die Variable frei, also einfach nur ‚pw=““ ‘ , da die Zugehörigkeit der Variable am Ende zufällig generiert werden muss.

Nun werden Alle Variablen initialisiert:

```
# Alle Variablen initialisieren!!!
gross = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
klein = "abcdefghijklmnopqrstuvwxyz"
sonderzeichen = "!""#""$%&'()*+,-./:;<=>?@{|}~"
zahlen = "123456789"
```

Alle Variablen, die man braucht wie beispielsweise Großbuchstaben, Kleinbuchstaben, Sonderzeichen und Zahlen werden nun initialisiert mit der genauen und nötigen Zugehörigkeit der Variable. Die Variablen sind die Bestandteile des Passwortes.

Der Benutzer darf nun die Passwortlänge eingeben:

```
laenge = int(input("Willkommen im Passwortgenerator!"
                   "\nBitte gebe die Passwortlänge ein!: "))
```

Nachdem die Variablen initialisiert wurden sind, darf der Benutzer seine gewünschte Länge für seinen zufälligen Passwort eingeben. Also haben wir eine Variable Länge initialisiert mit dem Namen „laenge“ und die Zugehörigkeit davon ist eine Input()-Funktion, wo der Benutzer selbst was auf der Konsole eintragen darf, in dem Fall ist es die Länge.

Jetzt werden die verschiedenen Optionen, Fragenartig für den Benutzer zur Verfügung gestellt:

```
frage = input(
    "Möchtest du Großbuchstaben,Kleinbuchstaben,Zahlen und Sonderzeichen ,dann drücke die 1.,"
    "\nwenn du Kleinbuchstaben, Zahlen und Sonderzeichen willst, dann drücke die 2.,"
    "\nwenn du Zahlen und Sonderzeichen willst, dann drücke die 3.,"
    "\nwenn du nur Sonderzeichen willst, dann drücke die 4.,"
    "\nwenn du nur Großbuchstaben willst, dann drücke die 5.,"
    "\nwenn du nur Kleinbuchstaben willst, dann drücke die 6. ,"
    "\nwenn du nur Zahlen willst, dann drücke die 7. ,"
    "\nwenn du Sonderzeichen und Großbuchstaben willst, dann drücke die 8.!"
    "\nwenn du Sonderzeichen und Kleinbuchstaben willst, dann drücke die 9.!"
    "\nwenn du Sonderzeichen und Zahlen willst, dann drücke die 10.!"
    "\nwenn du nur Großbuchstaben und Kleinbuchstaben willst, dann drücke die 11.,"
    "\nwenn du nur Großbuchstaben und Zahlen willst, dann drücke die 12.,"
    "\nwenn du nur Kleinbuchstaben und Zahlen willst, dann drücke die 13. ,"
)
```

Jetzt wird mit der Input Funktion dem Benutzer die Möglichkeit gegeben eine Option aus 13 verschiedenen Fragen aus zu suchen . Alle Optionen haben eine eigene Zahl, also kann der Benutzer sich eine Zahl welche ihn anspricht auswählen.

Die Auswahl der Option von dem Benutzer wird bearbeitet:

```
if frage == "1":
    for _ in range(laenge):
        pw = pw + secrets.choice(gross + klein + sonderzeichen + zahlen)
elif frage == "2":...
elif frage == "3":
    for _ in range(laenge):
        pw = pw + secrets.choice(sonderzeichen + zahlen)
elif frage == "4":
    for _ in range(laenge):
        pw = pw + secrets.choice(sonderzeichen)
elif frage == "5":
    for _ in range(laenge):
        pw = pw + secrets.choice(gross)
elif frage == "6":
    for _ in range(laenge):
        pw = pw + secrets.choice(klein)
elif frage == "7":
    for _ in range(laenge):
        pw = pw + secrets.choice(zahlen)
elif frage == "8":
    for _ in range(laenge):
        pw = pw + secrets.choice(gross + sonderzeichen)
elif frage == "9":
    for _ in range(laenge):
        pw = pw + secrets.choice(sonderzeichen + klein)
elif frage == "10":
    for _ in range(laenge):
        pw = pw + secrets.choice(zahlen + sonderzeichen)
elif frage == "11":
    for _ in range(laenge):
        pw = pw + secrets.choice(gross + klein)
```

Nun werden die 13 If-Abfragen je nach Eingabe bearbeitet und zusammengestellt. Es kommt drauf an welche Zahl der Benutzer von 1-13 ausgewählt hat und somit stellt das Programm die Auswahl zusammen. In jeder if-Abfrage befindet sich die am Anfang vom Benutzer eingegebene Länge, mit einer „for _ in range()“ Funktion. Diese Funktion hilft dabei, dass nur genauso viele Zeichen ausgegeben werden wie auch der Benutzer eingegeben hat. Dann wird die Variable pw mit den ausgewählten Zeichen zusammengetan.

Master Passwort

Wir haben ein Masterpasswort erstellt, in dem der Benutzer die Option bekommt ein neues Masterpasswort zu erstellen, wenn er noch keinen hat und wenn der Benutzer schon ein bestehendes Passwort besitzt, dann wird er nach diesem gefragt. Er hat 3 Versuche, um das richtige Passwort einzugeben andernfalls wird er nach drei fehlgeschlagenen Versuchen gebeten sich an den Support zu wenden und in Folge dessen wird das Programm beendet.

Masterpasswort „mpw“ definieren:

```
def mpw():
    leer = os.path.getsize("mpw.txt")
    if leer == 0:
        file = open("mpw.txt", "a")
        passwort = input("Gib dein neues Masterpasswort ein: ")
        file.write(passwort)
        file.close()
        menu()
```

Als erstes muss man eine Datei für das Masterpasswort im Voraus schon anlegen in diesem Falle ist es „mpw.txt“, dann wird geschaut, ob die Datei leer ist, wenn die Datei leer sein sollte, also „if leer == 0“, dann soll die Datei geöffnet werden mit „file = open“ und es soll das neue Passwort in die Datei reinschreiben dafür steht das „a“. Anschließend öffnet sich das Menü.

Wenn in der Datei ein Masterpasswort enthalten ist:

```
else:
    i=3
    with open("mpw.txt", "r") as f:
        for row in f:
            if row.split(","):
                frag = input("Was ist mpw: ")
                if frag == row:
                    menu()
                else:
                    while frag != row and i != 0:
                        print("Flasches Passwort! Noch ", i, "Versuche übrig.")
                        frag = input("Was ist mpw: ")
                        i = i-1
                if frag == row:
                    menu()
```

Wenn sich in der Datei schon ein Masterpasswort befindet, dann muss der Benutzer dieses richtig eingeben. Ihn stehen 3 Versuche zu Verfügung, dafürsteht „i=3“. Das System öffnet „mpw.txt“ und schaut in die Datei nach dem Passwort. Dafür steht das „r“ (read). Daraufhin wird nach dem Passwort gefragt. Wenn dieses richtig eingegeben worden ist, dann wird der Benutzer in das Menü geleitet. Wenn es falsch eingegeben worden ist, wird gesagt, dass es

falsch ist und mit der „ for-schleife „i“ “ verdeutlicht, wie viele Versuche der Benutzer noch frei hat. Und dann wird wiederholt nach dem Passwort gefragt.

Wenn das Passwort drei Mal falsch eingegeben worden ist:

```
if frag == row:
    menu()
else:
    print("Wende dich bitte an den Support!")
    sec = 5
    while sec != 0:
        sec = sec - 1
        time.sleep(1)
        print(sec)
    exit()
```

Wenn das Passwort dreimal falsch eingegeben wurde, dann wird vom System ausgesagt, dass man sich an den Support wenden soll. Ein Zeitlimit von 5 Sekunden wird durchlaufen und wird daraufhin beendet. Dies funktioniert mit der Whileschleife, solange die Schleife nicht null ist, zählt es runter, bis es null wird und dann geht es komplett raus aus dem System

Der Timer

Der Timer kann eine bestimmte Anzahl an Sekunden im Hintergrund zählen während andere Funktionen bedient werden können. Um das zu erreichen haben wir das „threading“ von Python benutzt. Sobald der Timer abläuft, wird die Funktion beendet und damit sind keine Aktionen mehr im Programm möglich d.h. es muss beendet werden. Möchte man den Passwort-Manager weiter benutzen, muss man das Programm neu starten und das Masterpasswort eingeben. Ansonsten gibt es keine Möglichkeit den Timer zu umgehen man muss das Programm neustarten.