Passwort-Manager

Aidai Almaz kyzy,
Dinara Kurmanbek,
Aikan-Suita Temirbaeva

Studiengang Wirtschaftsinformatik (IBIS)
Betriebssysteme & Rechnernetze
Frankfurt University of Applied Sciences
Nibelungenplatz 1
60318 Frankfurt am Main

Inhaltsverzeichnis

1. Einleitung

Die folgende Dokumentation zeigt die Entwicklung, als auch die Implementierung eines Passwort-Manager Programms auf der Kommandozeile mit Python auf. Ein Passwort-Manager funktioniert im Prinzip wie ein Tresor, da man beliebig viel Einträge speichern kann. Man muss nicht befürchten, sie zu vergessen. Der Manager ermöglicht auch eine sichere und eindeutige Erstellung eines Passworts für alle Webseiten. Die Benutzer können die Passwörter beliebig viel generieren lassen, kopieren, ins entsprechende Feld einfügen, ändern und löschen. Dazu wird das gesamte Programm auch mit einem Passwort gesichert.

Während der Projektarbeit stellten sich solche Probleme heraus:
1. Bei der Durchführung einer Aufgabe mit der Änderung des Master-Passworts standen wir vor dem Problem, dass der von uns geschriebene Code nicht das Master-Passwort löschte, sondern die ganze Datenbank.
2. Das Programm hat die fehlerhaften Dateneingaben des Benutzers nicht berücksichtigt, z. B. wenn die Passwortlänge als Buchstabe angegeben ist.
3. Bei einer fehlerhaften Dateneingabe wurde das Programm zunächst abgebrochen und beendet mit *exit()* und verhinderte damit, dass die korrekten Daten erneut eingegeben werden können.
4. Fehlt eine weitere Funktion.
5. Nicht vollständig ist die Aufgabe 8 bearbeitet, wo die Passwörter nicht im Klartext in der Tabelle angezeigt werden sollen.

2. Konzept

Für dieses Projekt wurde mit Hilfe **DB Browser for SQLite** eine globale Datenbank „passman.db“ angelegt (Abbildung 1). In Datenbank „passman.db“ wurden folgende Attribute hingelegt: db_name – PRIMARY KEY, master_pw und dt (datetime) (Abbildung 2). In der globalen Datenbank „passman.db“, wird eine Datenbank durch den Befehl (create-database) erstellt. Fürs Programm spielt die Reihenfolge der Eingabe in der Kommandozeile eine wichtige Rolle. Um das Programm zu verwenden, folgen Sie die Anweisungen unten: python3 passman.py create-database mydatabase(beliebiger Name der Datenbank).
Bei der Erstellung dieser Datenbank wird auch dazu ein Master-Passwort initialisiert und bei jeder nachfolgenden Funktion wird Master-Passwort wiederholt abgefragt. Es kann auch mit Hilfe der Anwendung (change) später

geändert werden. Parallel wird in neu erzeugter Datenbank eine Tabelle für die Einträge mit Attributen: TITLE, USERNAME und PASSWORD hinterlegt. Attribut Passwort kann vom Benutzer selbst angegeben werden oder vom Programm mit dem Befehl (generate) generiert. Das vorgeschlagene Passwort kann mit Zustimmung des Benutzers in Zwischenablage für weitere Manipulation gespeichert werden. Mit dem Befehl (add) kann man die neuen Einträge mit eigenem oder vom Programm generierten Passwort in eine Tabelle hinzufügen (Abbildung 3).

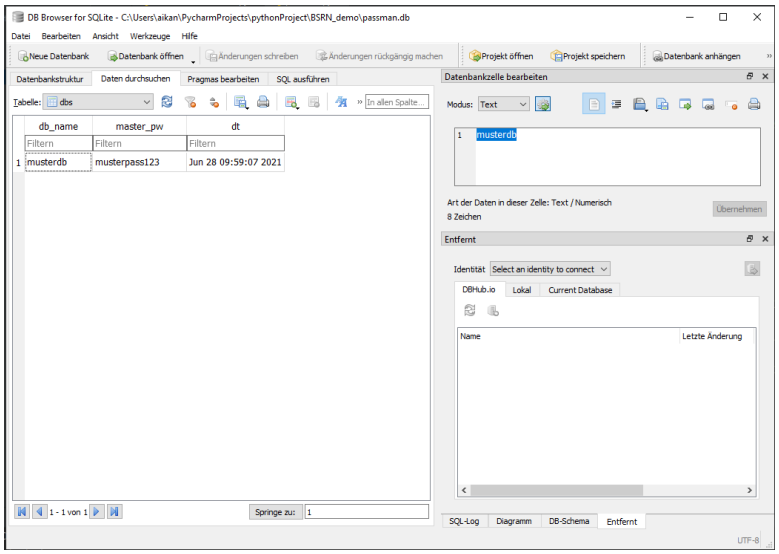

Abbildung 1. DB Browser for SQLite „passman.db“ mit Muster Datenbank

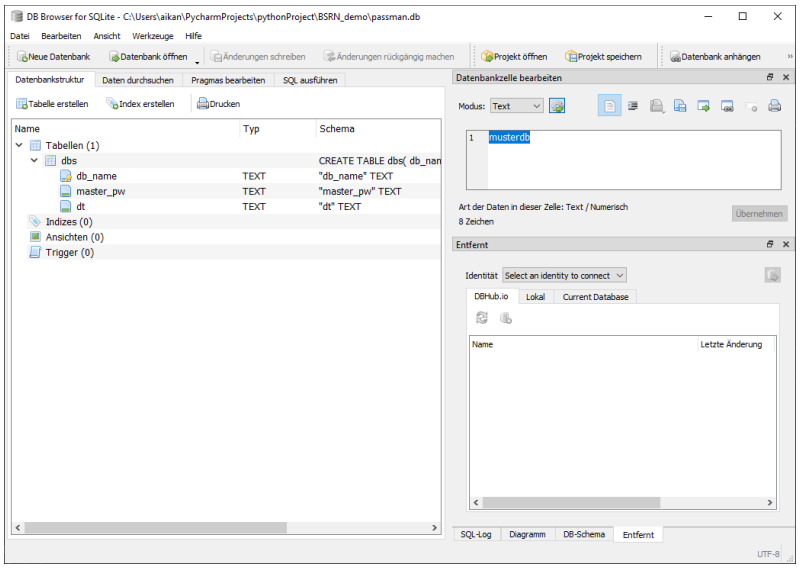

Abbildung 2. DB Browser for SQLite – Datenbankstruktur von „passman.db“

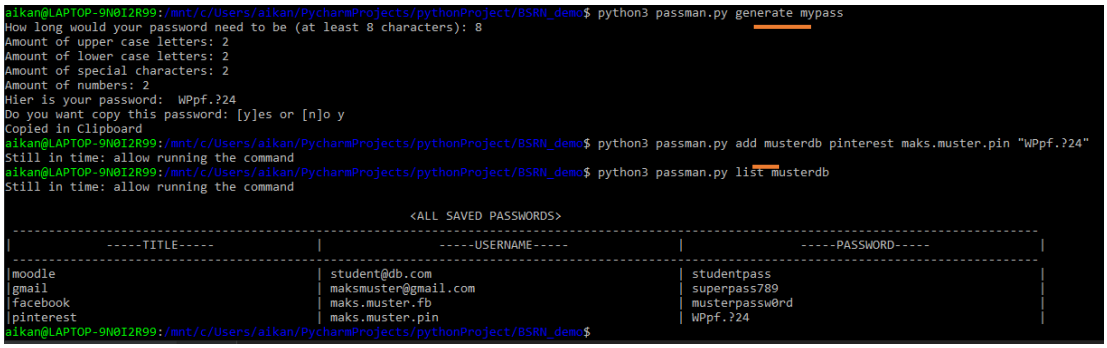

Abbildung 3. Befehlszeilenargumente „generate“ & „add“

3. Implementierung

Das Passwort-Manager Programm begann mit einer Problemstellung, z.B. die aufgestellten Anforderungen zuerst einzeln zu bearbeiten. Darauf folgte die Analyse des Problems z.B. die Machbarkeit herauszufinden, wo und wie werden die Daten gespeichert, die bearbeitete Funktionen zusammenzubringen. Nach diesem Schritt wurde ein Plan für Implementierung aufgestellt, der weiter auch umgesetzt wurde (Abbildung 4).

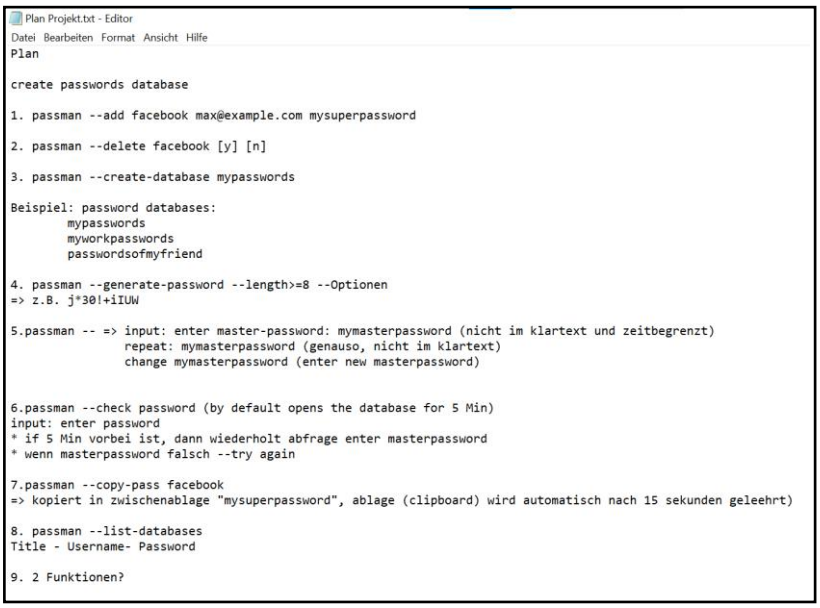

Abbildung 4. Plan des Projekts

4. Funktionen

Master-Passwort Programm enthält mehrere Hilfsfunktionen und Methoden, die im nächsten Abschnitt ausführlich dargestellt werden. Hilfsmethoden prüfen bestimmte Parameter, die für Weiterverarbeitung notwendig sind.

4.1. Hilfsmethoden

check_doable()
Diese Hilfsmethode wurde in jedem nachfolgenden Befehl aufgerufen, um die verbleibende Zeit für die Verwendung des Programms vom Benutzer zu überprüfen. Es wird Differenz zwischen dem aktuellen Datum/Uhrzeit (mit *datetime.now()*) und mit dem Zeitpunkt der letzten Passworteingabe berechnet, wenn 5 Minuten vergangen sind, wird die nächste **ask_password()-**Funktion aufgerufen, ansonsten läuft das Programm weiter.

ask_password()
Diese Hilfsmethode wird nur ausgeführt, wenn der Benutzer eine Zeitüberschreitung hat. Das Mater-Passwort wird abgefragt und bei Bestätigung des Master-Passwortes wird der Zeitpunkt der letzten Passworteingabe aktualisiert (mit *cur.execute()*), andernfalls fordert das Programm den Benutzer auf, das Passwort erneut einzugeben, bis das richtige Passwort angegeben ist (*while Schleife)*

password_parameters()
Diese Funktion fragt den Benutzer nach allen Optionen: Länge, Groß-/Kleinschreibung, Sonderzeichen/Symbole, die zum Generieren eines Passworts erforderlich sind. Es ist wichtig, dass der Benutzer diese Optionen in Zahlen eingibt, mit *While-Schleifen* und *try except-Konstrukten* werden alle Fehlereingaben berücksichtigt. Das Programm setzt die erforderliche Länge des Passworts auf 8 oder mehr Zeichen. Bei der richtigen Eingabe aller Optionen, wird ein Passwort, das mit Hilfe der Methode **password_generator()** generiert wird, vorgeschlagen.

password_generator()
Basierend auf den angegebenen Optionen sammelt diese Hilfsfunktion ein zufälliges (*random.choise(), join()*-Methoden *in for-Schleife*) Passwort, dafür wurden verschiedene char-Variablen in **password_generator()** initialisiert, die die Großbuchstaben, Kleinbuchstaben, Zahlen und Sonderzeichen enthalten.

4.2. Befehlszeilenargument

Die folgende Argumente: operation = sys.argv[1], db_name = sys.argv[2], title = sys.argv[3], username = sys.argv[4], password = sys.argv[5] sind im Programm

Passwort-Manager enthalten. Fürs Programm spielt die Reihenfolge der Eingabe von Argumenten in der Kommandozeile eine wichtige Rolle. Die korrekten Eingabemöglichkeiten für bestimmte Operationen werden unten erläutert.

4.2.1. Befehle

‚create-database'
Korrekte Eingabemöglichkeit: *python3* *passman.py* *create-database* *mydatabase* (beliebiger Name der Datenbank)

Es wird empfohlen, mit diesem Befehl zu beginnen, wenn keine Datenbank in **DB Browser for SQLite** für Einträge vorhanden ist. Bei der Verwendung dieses Befehls wird eine interne Datenbank erstellt und das Master-Passwort zweimal abgefragt, damit wird Übereinstimmung des Passworts geprüft. Wenn das Master-Passwort stimmt, wird in neu erzeugter Datenbank eine Tabelle für die Einträge mit Attributen: TITLE, USERNAME und PASSWORD hinterlegt, ansonsten wird die Datenbank nicht erstellt.

‚change'
Korrekte Eingabemöglichkeit: *python3* *passman.py* *change* *mydatabase*

Dieser Befehl ermöglicht das zuvor initialisierte und in Datenbank gespeicherte Master-Passwort zu ändern. Bevor der Benutzer das Master-Passwort ändert, wird er vom Programm aufgefordert, das alte Master-Passwort einzugeben und dann kann er ein neues eingeben. Nach der Eingabe des neuen Passworts wird Master-Passwort in DB aktualisiert.

‚generate'
Korrekte Eingabemöglichkeit: *python3* *passman.py* *generate* *password* (*der Name muss geschrieben werden, aber wie es benannt wird, ist dem Benutzer überlassen)

Dieser Befehl nimmt mit der Hilfsmethode **password_parameters()** generiertes Passwort und gibt es auf Kommandozeilenfenster aus. Der Benutzer hat die Möglichkeit dieses Passwort in Zwischenablage zu speichern und das Einfügen des Passworts geschieht mit dem Klick der rechten Maustaste in Kommandozeile (unsere 1.Funktion).

‚add'

Korrekte Eingabemöglichkeit: *python3 passman.py add mydatabase Moodle (Titel) MaxMustermann (Username) „superpass“(password)*

Diese Funktion fügt in die Tabelle neue Einträge hinzu und speichert sie in der Datenbank. In die Tabelle können nicht zwei Einträge mit gleichem Titel gespeichert werden, denn der Titel ist in Datenbank als PRIMARY KEY erstellt (Abbildung 5). Bei der Passworteingabe muss es in Anführungszeichen gesetzt werden, da einige Zeichen vom Programm nicht erkannt werden.

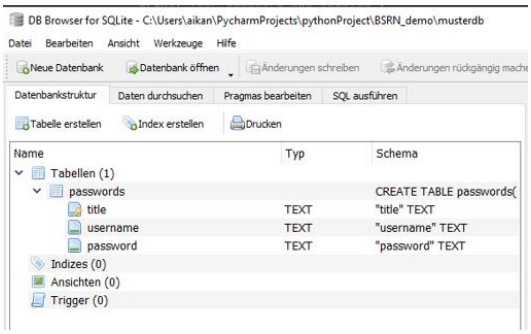

Abbildung 5. DB Browser for SQLite – Datenbankstruktur von „musterdb“

‚list‘
Korrekte Eingabemöglichkeit: *python3 passman.py list mydatabase*

Dieser Befehl gibt dem Benutzer die Möglichkeit alle seine Einträge aufzulisten. Die Einträge werden dann als eine Tabelle in der Shell ausgegeben.

‚delete‘
Korrekte Eingabemöglichkeit: *python3 passman.py delete mydatabase Moodle*

Mit Hilfe dieses Kommandos kann der User gespeicherte Einträge von der Tabelle/Datenbank löschen, aber davor wird einmal interaktiv um die Bestätigung nachgefragt.

‚get‘
Korrekte Eingabemöglichkeit: *python3 passman.py delete mydatabase Moodle*

Der Befehl kopiert das Passwort des Eintrags aus der Datenbank in Zwischenablage und man kann es z.B. in einer Webseite einfügen. Dafür ist die

Funktion *pyperclip.copy(„“)*. *time.sleep()*-Funktion verzögert das Programm für 15 Sekunden, nach dieser Zeit wird die Zwischenablage geleert.

5. Module und Bibliotheken

In diesem Abschnitt werden alle importierte Python-Module aufgelistet, die für das Master-Passwort Programm nötig sind.

import sys
Modul bietet Zugriff auf beliebige Befehlszeilenargumente über die sys.argv.
sys.argv ist die Liste der Befehlszeilenargumente.

import sqlite3
SQLite ist eine Bibliothek, die eine festplattenbasierte Datenbank bereitstellt, und den Zugriff auf die Datenbank ermöglicht.

import random
Dieses Modul implementiert Zufallfunktion für verschiedene Verteilungen.

import time
mit Hilfe dieses Modul bearbeitet man zeitbezogene Aufgaben.

import pyperclip
es ist ein plattformübergreifendes Python-Modul zum Kopieren und Einfügen von Zwischenablagefunktionen.

from datetime import datetime
datetime zuerst als Verweis auf die Klasse festlegen und dann sofort als Verweis auf das Modul, um mit Datumsangaben als Datumsobjekte zu arbeiten.

import getpass
Das Python-Modul ermöglicht es, den Benutzer zur Eingabe eines Passworts aufzufordern, ohne es auf dem Bildschirm anzuzeigen.

6. Ergebnisse von der Problemlösung

Die oben in der Einleitung genannten Probleme wurden wie folgt gelöst:

1. Für die Änderung des Master-Passworts wurde der Befehl ‚change' implementiert (Abbildung 6). Die bisherige Codesyntax war nicht richtig zusammengesetzt, so dass der Code nicht korrekt funktionierte. Deswegen dieses Teil des Codes wurde gelöscht und von Anfang neu geschrieben.

```
if operation == 'change':
    check_doable(db_name)
    while True:
        old_pass = getpass.getpass("Enter your old master password: ")
        conn = sqlite3.connect(passman_db)
        cur = conn.cursor()
        cur.execute("SELECT * FROM dbs WHERE db_name=?", (db_name,))
        result = cur.fetchone()
        if result[1] != old_pass:
            print("Wrong password...Try again")
        else:
            new_pass = getpass.getpass("Enter your new master password: ")
            conn = sqlite3.connect(passman_db)
            cur = conn.cursor()
            cur.execute("SELECT * FROM dbs WHERE db_name=?", (db_name,))
            result = cur.fetchone()
            if result[1] != new_pass:
                print('Your password was updated')
                cur.execute("UPDATE dbs SET  master_pw= ?", (new_pass,))
                conn.commit()
                break
            else:
                print('New password is same as your old password')
                exit()
```

```
if operation == "delete-masterpass":

    conn = sqlite3.connect(passman_db)
    cur = conn.cursor()
    cur.execute("UPDATE dbs SET master_password = '' WHERE name=?", (db_name,))
    print("Your Master Password is Deleting")

    new_masterpass1 = input("enter master password: ")
    new_masterpass2 = input("enter master password: ")
    if new_masterpass1 != new_masterpass2:
        print("passwords are not equals,aborting...")
        exit()
    conn = sqlite3.connect(passman_db)
    cur = conn.cursor()
    #cur.execute("INSERT or IGNORE INTO dbs VALUES(?,?,?);", (new_masterpass1))
    dt = datetime.now().strftime("%b %d %H:%M:%S %Y")
    cur.execute("INSERT or IGNORE INTO dbs VALUES(?,?,?);", (db_name, new_masterpass1, dt))
    conn.commit()
```

Abbildung 6. Operation ←(neu) (old)→‚change' und ‚delete-masterpass'

2. Um die fehlerhaften Dateneingaben des Benutzers zu berücksichtigen, wurde *try-except* Konstruktor hinzugefügt, der die Fehlereingabe an den Benutzer meldet. Und es wird auch unendliche While-Schleife durchgespielt, bis der richtige Parameter eingegeben wird (Abbildung 7).

```
def password_parameters():
    while True:
        password_length = input("How long would your password need to be (at least 8 characters): ")  # 8
        if password_length != int:
            try:
                if int(password_length) >= 8:
                    password_length1 = int(password_length)
                    break
            except ValueError:
                print("Parameters need to be an digit, try again...")
```

Abbildung 7. password_parameters()-Funktion

3. Bei einer fehlerhaften Dateneingabe wurde das Programm zunächst abgebrochen und beendet und um das zu verhindern, wurden bei der Funktion *ask_password()* folgende Änderungen vorgenommen: vor der Passwortabfrage wurde eine *While-Schleife* eingefügt, dies löste das Problem des Programmstopps bei falscher Passworteingabe. Das *exit()*-Kommando wurde ebenfalls entfernt und somit fragt das Programm nach einem Passwort, bis es

richtig eingegeben wird. Wenn Master-Passwort richtig eingegeben wird, stoppt die *While-Schleife* mit dem break-Operator (Abbildung 8).

```
def ask_password(db_name):
    master_password = raw_input("enter master password: ")
    conn = sqlite3.connect(passman_db)
    cur = conn.cursor()
    cur.execute("SELECT * FROM dbs WHERE db_name=?", (db_name,))
    result = cur.fetchone()
    if result[1] == master_password:
        print('password correct, updating timestamp')
        dt = datetime.now().strftime("%b %d %H:%M:%S %Y")
        cur.execute("UPDATE dbs SET dt = ?", (dt,))
        conn.commit()
    else:
        print('not correct, aborting...')
        exit()
```

```
def ask_password(db_name):
    while True:
        master_password = getpass.getpass("Repeat your master password: ")
        conn = sqlite3.connect(passman_db)
        cur = conn.cursor()
        cur.execute("SELECT * FROM dbs WHERE db_name=?", (db_name,))
        result = cur.fetchone()

        if result[1] == master_password:
            print('Password correct, updating timestamp.')
            dt = datetime.now().strftime("%b %d %H:%M:%S %Y")
            cur.execute("UPDATE dbs SET dt = ?", (dt,))
            conn.commit()
            break
        else:
            print('Password is wrong, aborting...')
```

Abbildung 8. Funktion ask_password() ←(old)(new)→

4. Im Programm ist nur die Eingabe des Master-Passworts nicht im Klartext angezeigt, aber die Passwörter von den Einträgen in der Tabelle sind leider nicht verschlüsselt.

7. Fazit

Das Thema für das Projekt wurde von uns gewählt, weil das Thema Passwort-Manager aus dem Modul OOP bekannt war. Da das Projekt jedoch nicht in Java geschrieben werden konnte, mussten wir für die Erstellung des Programms unbekannte Programmiersprache Python selbstbeibringen, was die meiste Zeit in Anspruch nahm. Beim Aufbau des Projekts waren wir mit vielen Hindernissen konfrontiert, von denen wir viele lösen konnten, aber einige sind doch offengeblieben. Auf der anderen Seite war es für unsere Gruppe eine großartige Lernkurve, über die wir sehr froh sind. Wir haben eine Menge gelernt, zum Beispiel ist die Art und Weise, wie man Code in Python schreibt, anders als in Java. Wir waren uns auch einig, dass Python einfacher zu bedienen ist als Java.

8. Anhang

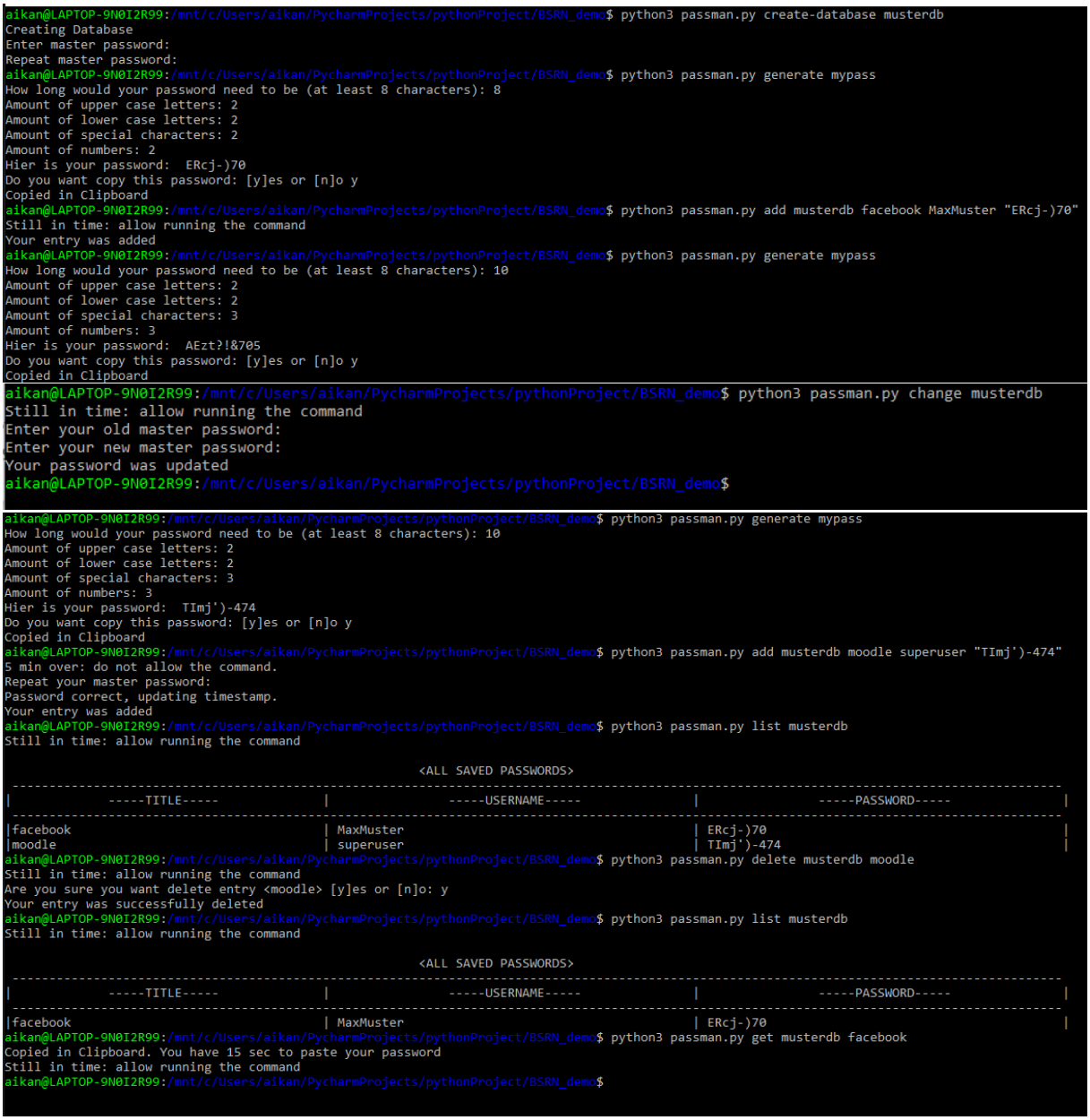

Abbildung 1. Komplette Ausführung des Programms im Ubuntu Terminal