BS&RN – Dokumentation

Passwort-Manager

Werkstück A

Alternative 4

Studiengang „International Business Information Systems“

Modul „Betriebssysteme und Rechnernetze“

Prof. Dr. Christan Baun

Frankfurt University of Applied Sciences

Von:

Florian Harms (1260386)

Nils Kozonek (1085545)

Inhaltsverzeichnis

1. Einleitung

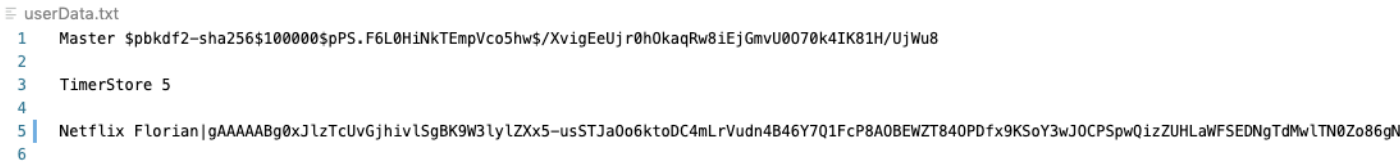
Durch die stetige Nutzung vieler Internetplattformen, wie auch verschiedenster Applikationen, sind die Nutzer zumeist dazu angehalten sich zu registrieren. Dadurch fallen selbstverständlich so manche Nutzernamen, wie auch Passwörter an, die aus Sicherheitsgründen möglichst komplex, lang und im besten Fall noch unterschiedlich sind. Deshalb nutzen nunmehr knapp ein Viertel aller Leute einen Passwort-Manager.
In dieser Ausarbeitung zeigen wir auf, wie wir unseren Passwort-Manager in Python konzipiert haben und gehen näher auf die grundlegenden Funktionen, und deren Nutzen, sowie auf unsere zwei weiteren Funktionen, das Passworthashen, wie auch auf das Passwortverschlüsseln ein.

2. Erklärung unserer Ideen

In der Aufgabenstellung der Alternative 4 sollen zwei weitere sinnvolle Funktionen implementiert werden. Hierfür haben wir uns für das Hashen und die Verschlüsselung der Passwörter entschieden. Bei Passwörtern handelt es sich um sehr sensible Daten. Diese Daten zu sichern und möglichst gut zu verschlüsseln war für uns unbedingt notwendig. Um das Programm noch sicherer zu machen, haben wir uns zusätzlich für das Hashen der Passwörter mit Hilfe einer Hash Funktion entschieden.

3. Erklärung des Lokalen Speichers

Alle Daten werden in einer Textdatei mit dem Namen userData.txt gespeichert. Diese Datei befinden sich in dem gleichen Ordner wie unsere Python Dateien mit dem Quellcode. Die Textdatei enthält drei Arten von Daten, die jedoch alle in der gleichen Form gespeichert sind, nämlich nach dem Titel.
Die erste Art von Daten ist das Master Passwort. Diese Daten sind in Form des Titels, welcher Master lautet und der Daten, welche eine gehashte Version des Master Passwortes beinhalten.
Die zweite Art von Daten ist die Timeout Dauer, da dies vom Benutzer gewählt wird, muss sie lokal gespeichert werden. Diese Daten werden in der Form des Titels, welcher TimerStore lautet und der Daten in Form der Timeout Dauer in Minuten angegeben.
Der letzte Datentyp sind die Daten der Benutzernamen und Passwörter der Services, die der Benutzer gespeichert hat. Diese werden in Form des Titels, mit dem Namen der Services und den Daten in Form von Benutzername und verschlüsseltes Passwort angegeben.



4. Erklärung der Timeout Funktion

In Python wird der Code sequentiell ausgeführt. Dies bedeutet, dass das Erstellen einer Funktion, die einen Benutzer nach einer bestimmten Zeit wieder abmeldet, sehr Herausfordernd sein kann. Zur Hilfe gibt es allerdings ein „Timer“ Objekt, welches wir verwenden können.
Das Programm funktioniert, indem der Callback Funktion ein Array mit dem Namen timeoutMessage zugewiesen wird. Dieses Array wird überall im Programm überprüft, wo der Benutzer eine Entscheidung trifft. Nach Ablauf des Logout Timers ist die timeoutMessage überall im Programm zu sehen und der Benutzer wird von der Interaktion mit der Anwendung abgehalten und gezwungen sich wieder anzumelden.

5. Hashfunktion

Eine Hash-Funktion ist eine one-way-Funktion. Dieser Funktion wird ein String übergeben und im Anschluss gibt die Funktion einen gehashten String zurück. Hierbei ist zu erwähnen, dass es unmöglich ist, den gehashten String wieder in seine ursprungsform zu verwandeln.

„Mein Passwort -> Hash Funktion ->
$pbkdf2sha256$100000$xrhX6n2PUUpJ6Z0Twtib8w$xDV8TchkOc03GOPXPTfWcH9zwc1CbU

Wird eine gehashte Version eines Passworts gespeichert, kann beim nächsten Anmeldeversuch eines Benutzers, dessen Eingabe durch dieselbe Hashing Funktion analysiert und geprüft werden. Sollten die beiden gehashten Strings miteinander übereinstimmen, wird das Passwort akzeptiert. Hierbei ist es also nicht notwendig, dass eigentliche Passwort zu speichern und es kann trotzdem überprüft werden, ob das richtige Passwort eingegeben wurde.
In unserem Programm wird Hashing dazu verwendet, das Master Passwort sicher zu speichern. Sollte das Master Passwort vom Benutzer vergessen werden, ist er allerdings für immer aus dem Programm ausgesperrt, da es wie schon erwähnt nicht möglich ist, den gehashten String umzukehren und zum Ursprungs Passwort zurückzukehren.

6. Verschlüsselung

Verschlüsselung ist ähnlich wie das Hashen, allerdings kann man hierbei den Prozess umkehren. Es ist möglich einen verschlüsselten String wieder zu entschlüsseln. Für die Verschlüsselung ist zudem ein eindeutiger Schlüssel erforderlich, der zum Verschlüsseln und Entschlüsseln von Daten verwendet wird. Im Vergleich zum Hashen funktioniert dies folgendermaßen:

„Mein Passwort“ -> Verschlüsselungsfunktion mit Schlüssel -> $pbkdf2sha256$100000$xrhX6n2PUUpJ6Z0Twtib8w$xDV8TchkOc03GOPXPTfWcH9zwc1CbU -> Entschlüsselungsfunktion mit Key -> „Mein Passwort“

Hierbei ist zu beachten, dass bei beiden Funktionen der gleiche Schlüssel verwendet wird, um das ursprüngliche Passwort wieder zu erhalten.
In unserem Programm wird ein Verschlüsselung Key mit dem Master Passwort erzeugt, wenn der Benutzer dieses eingibt. Dadurch ist die Textdatei mit allen Passwörtern gesichert, da im Falle eines Datendiebstahls, ein Schlüssel benötigt wird und die Passwörter zu entschlüsseln. Um an diesen Schlüssel zu gelangen, wird jedoch das Master Passwort benötigt. Aufgrund des gehashten Master Passworts ist es außerdem nicht möglich dies umzukehren, was das Programm ungemein sicher macht.

7. Erklärung des Quellcodes

7.1 Hash- und Verschlüsselungsfunktionen

```
21  #--------------------Hashing- und Verschlüsselungsfunktionen--------------------#
22
23
24  def hasher(mode, plainText, hashedPassword=None):
25      context = CryptContext(
26              schemes=["pbkdf2_sha256"],
27              default="pbkdf2_sha256",
28              pbkdf2_sha256__default_rounds=100000
29      )
30      if mode == 'hash':
31          return context.hash(plainText)
32      elif mode == 'check':
33          return context.verify(plainText, hashedPassword)
34
35  def generateKey(master):
36      salt = b'H\x1d\tMg\xc9\xe3\xec\xbeU\xee\x03\xec\x18\xf1U'
37      kdf = PBKDF2HMAC(
38          algorithm=hashes.SHA256(),
39          length=32,
40          salt=salt,
41          iterations=100000
42      )
43      return base64.urlsafe_b64encode(kdf.derive(master.encode()))
```

Hier finden wir einen Auszug aus unserem Quellcode. Die Hash Funktion aus Zeile 24 wird hier benutzt, um plainText zu hashen und Hashes miteinander zu vergleichen. Gleichzeitig wird die Klasse CryptContext() mit dem richtigen Algorithmus Satz erzeugt. Bei PBKDF2 handelt es sich um eine genormte Funktion, um von einem Passwort einen Schlüssel abzuleiten. Außerdem wird PBKDF2 auch häufig für Passwort basierte Authentifizierung verwendet. SHA256 stellt eine Variante der kryptografischen Hashfunktionen dar.
Die Funktion generateKey() erzeugt einen kryptografischen Schlüssel basierend auf einem gegebenen String und benutzt dafür eine Key Derivate Funktion, wie es Beispielsweise PBKDF2 ist.
Die Variable salt ist ein zufälliger String, der zur Sicherung des Schlüssels bei seiner Erstellung verwendet wird.
Zum Schluss wird die Kryptografische Funktion noch zurückgegeben.

7.2 UI Funktionen

```
45  #----------------------Ui Funktionen----------------------#
46  #Diese Funktionen sind in erster Linie dafür da, die main() Funktion clean zu halten.
47
48  def welcomeMessage():
49      print(
50  '''
51  _________________________________
52
53      Nils and Florian's Password Manager
54  _________________________________
55  ''')
56
57  def mainMenu():
58      selection = input(
59  '''Please select one of the following program functions:
60  1 - Get a list of all services for which there is stored username and password
61  2 - Save a new service's username and password
62  3 - Delete a saved service's username and password
63  4 - Change master password
64  5 - Change login timeout duration
65  6 - Fetch a service's username and password
66  7 - Close program
67  Type your selection here: ''')
68      return selection
69
70  def showServiceList():
71      print('List of stored services: ')
72      storedServices = getStoredData().keys()
73      for service in storedServices:
74          if service != 'Master' and service != 'TimerStore':
75              print(f' - {service}')
```

Im nächsten Schritt des Quellcodes, haben wir die Funktionen des User Interfaces beschrieben. Um die main() Funktion etwas klarer zu halten, haben wir die Statements separat implementiert. Zuerst wird der Benutzer durch die Nachricht „Nils and Florian's Password Manager" begrüßt. Die nächste Funktion empfängt den Benutzer im Haupt Auswahlmenü. Hier kann wie in der Aufgabenstellung beschrieben, zwischen den verschiedenen Programm Funktionen gewählt werden.
Mit der Funktion showServiceList(), wird eine Liste aller gespeicherten Services erzeugt und deren Titel ausgegeben. Hierfür werden auch nur die Titel benötigt, da die Schlüssel aus dem Verzeichnis gezogen werden, welches dann getStoredData() zurückgibt.
In unserem Verzeichnis sind allerdings auch das Master Password und der ‚TimeStore' gespeichert, welche in Zeile 73f. herausgefiltert werden.

7.3 Utility Funktionen

```
77  #----------------------Utility Funktionen----------------------#
78
79  def getStoredData():
80      directory = os.path.dirname(os.path.realpath(__file__))
81      file = open(directory +'/userData.txt', 'r')
82      dataDictionary = {}
83      for line in file:
84          if len(line.strip()) != 0:
85              key, value = line.split()
86              dataDictionary[key] = value
87      return dataDictionary
```

In diesem Teil des Quellcodes, wird die Archivierung und das Kopieren dieser Daten vorgenommen. Die Funktion getStoredData() liest die Textdatei userData und gibt ein Verzeichnis in der Form {'Titel': Benutzername|VerschlüsseltesPasswort'} zurück. Jede Zeile in diesem Verzeichnis hat die beschriebene Form. Die Methode .split(), teilt die Strings auf, sobald ein Leerzeichen vorhanden ist. Zum Schluss wird das dataDictionary wieder zurückgegeben.

7.4 Call Back Funktionen

```
89  #----------------------Call back Funktionen----------------------#
90  def passwordTimeout(timeoutMessage):
91      timeoutMessage.append('timeoutReached')
92
93  def clipboardTimeout():
94      copy('')
```

Callback Funktionen sind solche Funktionen, die nach einer bestimmten Zeit ausgeführt und dann später aufgerufen werden. Callback Funktionen eignen sich dazu, Funktionen erweiterbar zu machen und dann bei bestimmten Umständen oder Ereignisse aufgerufen zu werden.
Die Funktion passwordTimeout() fügt dem angegebenen Array ein weiteres Element hinzu. Weiterhin wird in Zeile 94 die Zwischenablage geleert.

7.5 Main Programm

```
96  #---------------------Main Programm Funktionen---------------------#
97
98  #Funktion zur Überprüfung des Master-Passworts eines Benutzers
99  def logOn():
100     userInput = getpass(prompt='Please enter the master password (characters typed will be hidden): ')
101     storedHash = getStoredData()['Master']
102     return hasher('check', userInput, storedHash), generateKey(userInput)
```

Als nächstes wird ausführlich auf das Main Programm unserer Anwendung eingegangen. Zuerst ist es wichtig, das Master Passwort welches der Benutzer beim Start des Programmes eingeben muss zu überprüfen.
Die Funktion getpass() funktioniert wie die Funktion input(), allerdings kann der Benutzer nicht sehen, was er gerade eingibt und bekommt dies auch dementsprechend angezeigt (characters typed will be hidden). Anschließend zieht sich die Funktion getStoredData() den gespeicherten Hash des Master Passworts.
In Zeile 102 findet nun eine Rückgabe der verglichenen Hashes, des Master Passworts und der Benutzereingabe statt. Außerdem wird der Verschlüsselungs Key der durch die Benutzereingabe generiert wurde, zurückgegeben.

Master Passwort ändern

```
125  #ChangeMasterKey erlaubt dem Benutzer das Maser Passwort zu ändern
126  def changeMasterKey(timeoutMessage, encryptionKey):
127
128      userInput = getpass(prompt='Please enter the new master password: ')
129      checkUserInput = getpass(prompt='Please re-enter the new master password: ')
130
131      if len(timeoutMessage) > 0:
132          print('Error: Your session has timed out, you will need to log in again')
133          return 'timeout'
134      elif userInput == checkUserInput:
135
136          storedData = getStoredData()
137
138          storedData['Master'] = hasher('hash', userInput)
139
140          newEncryptionKey = generateKey(userInput)
141          fernetOldKey = Fernet(encryptionKey)
142          fernetNewKey = Fernet(newEncryptionKey)
143          for title in storedData.keys():
144              if title != 'Master' and title != 'TimerStore':
145                  username, encryptedPassword = storedData[title].split('|')
146                  decryptedPassword =  fernetOldKey.decrypt(encryptedPassword.encode())
147                  newEncryptedPassword = fernetNewKey.encrypt(decryptedPassword).decode()
148                  storedData[title] = f'{username}|{newEncryptedPassword}'
149
150          directory = os.path.dirname(os.path.realpath(__file__))
151          with open(directory + '/userData.txt', 'w') as file:
152              for key in storedData.keys():
153                  print(f'{key} {storedData[key]}\n', file=file)
154          print('Master password successfully changed!')
155          return newEncryptionKey
156      else:
157          print('Error: Those passwords did not match')
158          return None
```

Weiter geht es mit der Funktion zum Ändern des Master Passwortes. In der Zeile 126f. wird der Benutzer dazu veranlasst, das Passwort zweimal einzugeben. Damit soll sichergestellt werden, dass das neue Passwort korrekt eingegeben wurde. Mit der Funktion getpass() geschieht dies wieder mit nicht sichtbaren Zeichen. In Zeile 131 muss nun durch eine If-Abfrage geprüft werden, ob in der Zeit, die der Benutzer für die Eingabe des neuen Passworts benötigt hat, keine Zeitüberschreitung erfolgt ist. Sind in Zeile 134 die beiden Passwörter identisch, dann wird der gespeicherte Hash des Master Passworts aktualisiert und es werden alle Passwörter neu verschlüsselt. Genauer beschrieben, wird in Zeile 138 das Verzeichnis mit dem neuen Hash des Master Passwortes aktualisiert und in den folgenden Zeilen werden alle Passwörter wieder verschlüsselt. In Zeile 150 wird anschließend die userData Datei mit dem aktualisierten Verzeichnis überschrieben. Stimmen die Passwörter in Zeile 156f. nicht überein, kehrt der Benutzer zum Hauptmenü zurück.

Einstellen des Timeouts

```
252  #Die Funktion erlaubt es dem Benutzer die Dauer des Anmelde Timeouts zu ändern
253  def changeLoginTimout(timeoutTimer, timeoutMessage):
254
255      timeoutChoice = True
256      while timeoutChoice:
257          newTimeout = input('Please type the number of minutes you would like the logout timer to be: ')
258          if len(timeoutMessage) > 0:
259              print('Error: Your session has timed out, you will need to log in again')
260              return 'timeout'
261          isInteger = False
262          try:
263              newTimeout = int(newTimeout)
264              isInteger = True
265          except:
266              print('You must enter an integer!')
267          if isInteger:
268              if newTimeout < 5 or newTimeout > 180:
269                  print('Please pick a time between 5 and 180 minutes.')
270              else:
271                  timeoutTimer.cancel()
272                  timeoutMessage = []
273
274                  storedData = getStoredData()
275                  storedData['TimerStore'] = newTimeout
276                  directory = os.path.dirname(os.path.realpath(__file__))
277                  with open(directory + '/userData.txt', 'w') as file:
278                      for key in storedData.keys():
279                          print(f'{key} {storedData[key]}\n', file=file)
280                  print('Timeout duration changed!')
281                  return Timer(newTimeout*60, lambda: passwordTimeout(timeoutMessage))
```

Mit der Funktion changeLoginTimout() hat der Benutzer die Möglichkeit die Zeit des Timeouts bei der Anmeldung zu ändern. Zuerst wird in dieser Funktion eine Schleife erstellt. In Zeile 257 kann dann vom Benutzer die Dauer in Minuten festgelegt werden, die als Timeout Zeit gewünscht ist. Hierbei kann selbst entschieden werden, wann das Passwort erneut eingegeben werden soll. In Zeile 261ff. wird dann geprüft, ob der Benutzer einen integer Wert eingegeben hat. Sollte dies der Fall sein, wird in Zeile 268 geprüft, ob der Timeout zwischen 5 Minuten und 3 Stunden liegt. Sind dann alle eingaben korrekt, wird in Zeile 270 ff. der alte Timer gelöscht und es wird ein neuer erstellt. Nun muss in Zeile 274 ff. noch der lokale Speicher aktualisiert werden, um die Änderung der Timeout Dauer zu berücksichtigen. Im letzten Schritt in dieser Funktion wird das neue Timeout Objekt mit return zurückgegeben und gleichzeitig auch genannt.

Löschen der gespeicherten Services

```
283  #Mit der Funktion kann der Benutzer einen gespeicherten Service löschen
284  def deleteService(timeoutMessage):
285
286      showServiceList()
287      choice = input('Please enter the title of the service you want to remove: ')
288      if len(timeoutMessage) > 0:
289          print('Error: Your session has timed out, you will need to log in again')
290          return 'timeout'
291      storedData = getStoredData()
292      storedTitles = [key for key in storedData.keys() if key != 'Master' and key !='TimerStore']
293      if choice in storedTitles:
294          storedData.pop(choice)
295          storedTitles.pop(storedTitles.index(choice))
296
297          directory = os.path.dirname(os.path.realpath(__file__))
298          with open(directory + '/userData.txt', 'w') as file:
299              for key in storedData.keys():
300                  if key != choice:
301                      print(f'{key} {storedData[key]}\n', file=file)
302          print(f'{choice} has been removed from the system')
303      else:
304          print(f'{choice} is not an option, you will now be taken to the main menu')
```

Mit der Funktion deleteServices() ist es dem Benutzer möglich, einen gespeicherten Service zu löschen. Die Funktion showServiceList() zeigt dem Benutzer zunächst die zur Auswahl stehenden Services an und lässt ihn im folgenden Auswählen, welcher gelöscht werden soll. In Zeile 291 f. sollen alle Service Titel in eine Liste geladen werden und anschließend wird durch eine if-Anweisung geprüft, ob die Auswahl des Benutzers in der Liste der Titel enthalten ist. Wird die Auswahl des Benutzers im Folgenden in der Liste gefunden, wird sie in Zeile 294 f. aus dem Verzeichnis entfernt. Anschließend wird die userData Datei in Zeile 297 mit den aktualisierten Daten überschrieben. Sollte der Benutzer generell keine gültige Eingabe gemacht haben, kehrt er durch Zeile 303 f. zum Hauptmenü zurück.

Benutzername und Passwörter abrufen

```
306  #mit dieser Funktion kann der Benutzer Benutzernamen und Passwörter abrufen
307  def fetchServiceDetails(timeoutMessage, encryptionKey):
308      showServiceList()
309      choice = input('Please enter the title of the service you want to view: ')
310      if len(timeoutMessage) > 0:
311          print('Error: Your session has timed out, you will need to log in again')
312          return 'timeout'
313      storedData = getStoredData()
314      storedTitles = [key for key in storedData.keys() if key != 'Master' and key !='TimerStore']
315      if choice in storedTitles:
316          username, encryptedPassword = storedData[choice].split('|')
317          fernet = Fernet(encryptionKey)
318          decryptedPassword =  fernet.decrypt(encryptedPassword.encode()).decode()
319          copy(decryptedPassword)
320
321          clipboardTimer = Timer(60,  clipboardTimeout)
322          clipboardTimer.start()
323
324          print(f'Your username for {choice} is: {username}, and your password has been coppied to the clipboard where it will remain for 1 minute.')
325          return clipboardTimer
326      else:
327          print(f'{choice} is not an option, you will now be taken to the main menu')
328          return None
```

Mit der Funktion fetchServiceDetails() ist es dem Benutzer möglich, die Benutzernamen und Passwörter abzurufen. Der erste Teil der Funktion (bis Zeile 314) ist nahezu identisch mit der zuvor beschriebenen Funktion zum Löschen der gespeicherten Services.
In Zeile 316 wird wie zuvor schon einmal beschrieben, der Benutzername und das Passwort mithilfe eines „|“ getrennt. Daraufhin wird mit dem Konstruktor Fernet ein Fernet Objekt mit dem Verschlüsselungs Key, abgeleitet vom Master Passwort erstellt.
Das Fernet Modul des Kryptographie Pakets von Python verfügt über eingebaute Funktionen zur Erzeugung eines Schlüssels, zur Verschlüsselung von Klartext in Ciphertext und zur Entschlüsselung von Ciphertext in Klartext mit den Methoden encrypt und decrypt. Das Fernet Modul garantiert, dass damit verschlüsselte Daten ohne den Schlüssel nicht weiter manipuliert oder gelesen werden können.
Anschließend wird in Zeile 319 ff. das entschlüsselte Passwort in die Zwischenablage kopiert und diese wird danach so eingestellt, dass sie in 60 Sekunden wieder gelöscht wird. Bei einer ungültigen Auswahl wird, der der Benutzer wieder darüber informiert und gelangt zurück zum Hauptmenü.

Main Funktion

```
330  #Main Funktion:
331  def main():
332
333      welcomeMessage()
334      clipboardUsed = False
335      running = True
336      while running:
337          logOnLoop = True
338          while logOnLoop:
339              valid, encryptionKey = logOn()
340              if valid:
341                  logOnLoop = False
342                  print('Welcome Back!')
343              else:
344                  print('Error: That is not the correct password')
345
346          storedTimeout = int(getStoredData()['TimerStore'])*60
347          timeoutMessage = []
348
349          timeoutTimer = Timer(storedTimeout, lambda: passwordTimeout(timeoutMessage))
350          timeoutTimer.start()
```

Im Folgenden wird noch einmal auf die Main Funktion unserer Anwendung eingegangen. Da die Willkommens Nachricht nur einmal angezeigt werden soll, wird sie außerhalb der laufenden Schleife aufgerufen. In Zeile 337 wird festgelegt, dass der Benutzer solange in der Anmeldeschleife hängt, bis er das richtige Master Passwort eingegeben hat. Weiter wird in Zeile 346 das gespeicherte Timeout lokal gespeichert, da der Benutzer die Dauer bearbeiten kann. Der Timer wird mit der Funktion passwordTimeout() laufen, nachdem die Dauer der variable storedTimeout erreicht ist. Durch die Funktion lambda ist es möglich, Argumente der angegebenen Funktion zu parsen, ohne dass diese läuft. Genauer gesagt ist eine Lambda Funktion eine kleine anonyme Funktion, die ein Objekt zurückgibt. Das von lambda zurückgegebene Objekt wird normalerweise einer Variablen zugewiesen oder als Teil anderer größerer Funktionen verwendet. Ein lambda ist besser lesbarer als eine vollständige Funktion, da es in-line geschrieben werden kann.

Quellen

- https://www.educative.io/edpresso/what-is-a-python-lambda-function?https://www.educative.io/courses/grokking-the-object-oriented-design-interview?aid=5082902844932096&affiliate_id=5082902844932096&utm_source=google&utm_medium=cpc&utm_campaign=grokking-ci&utm_term=&utm_campaign=Grokking+Coding+Interview+-+USA%2B&utm_source=adwords&utm_medium=ppc&hsa_acc=5451446008&hsa_cam=1871092258&hsa_grp=84009716779&hsa_ad=396821895536&hsa_src=g&hsa_tgt=aud-597782228586:dsa-1287243227899&hsa_kw=&hsa_mt=b&hsa_net=adwords&hsa_ver=3&gclid=Cj0KCQjw5uWGBhCTARIsAL70sLJWY1aLCZA_TAFZ0e104QIxin-em-P3aXpgRhS9gFFPvgAzza-CvzAaAkAUEALw_wcB

- https://www.geeksforgeeks.org/fernet-symmetric-encryption-using-cryptography-module-in-python/

- https://www.ionos.de/digitalguide/websites/web-entwicklung/was-ist-eine-callback-function/

- https://www.biteno.com/was-ist-sha256/

- https://py-tutorial-de.readthedocs.io/de/python-3.3/modules.html