Johannes Merkert

Schlangenbeschwörung für Einsteiger

Programmieren lernen mit Python

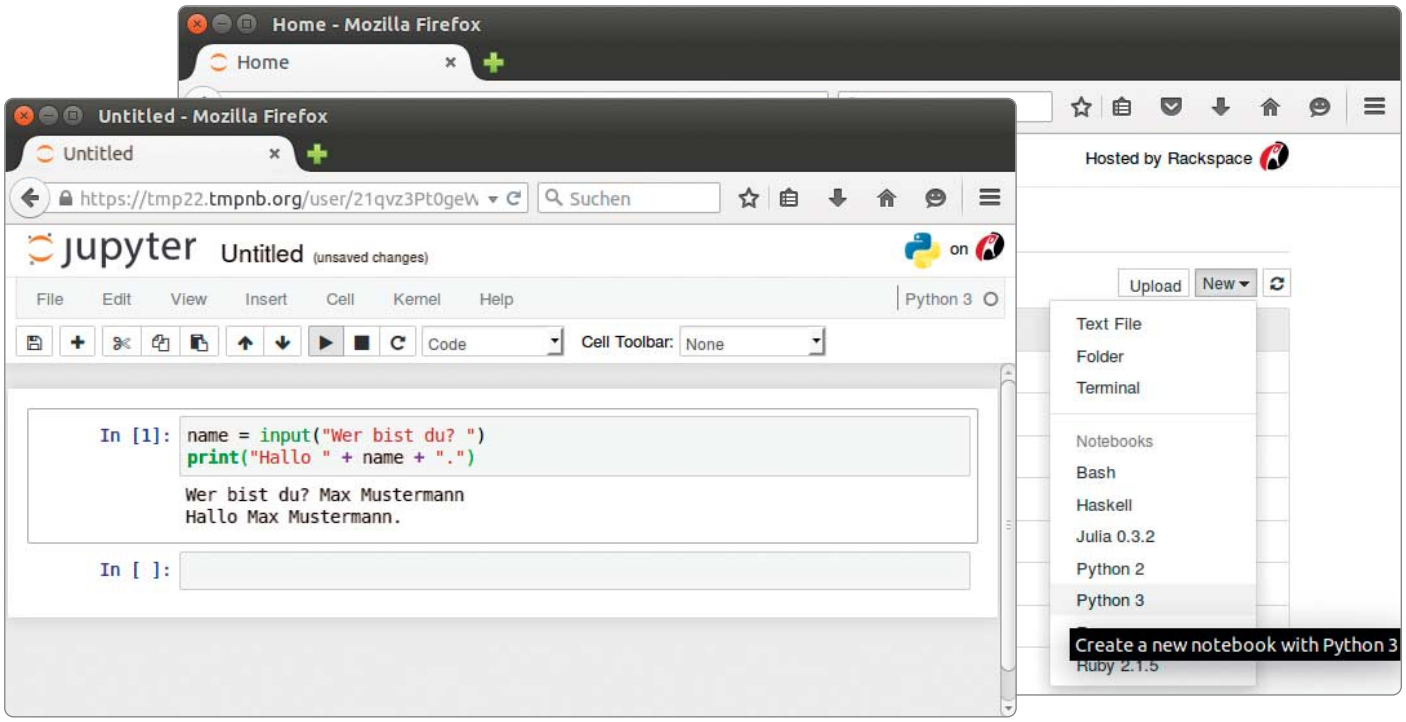
Programmierkenntnisse eröffnen am Computer neue Möglichkeiten. Damit der Rechner folgt, muss man aber erst mal dessen Sprache lernen. Python macht den Einstieg leicht: Wenige Zeilen Code reichen, um einen sicheren und alltagstauglichen Passwort-Manager auf die Beine zu stellen.

Für den Bau einer Kathedrale bedarf es Hunderter hoch spezialisierter Baumeister und viel Zeit. Bei komplexer Software ist es ähnlich: Niemand programmiert mal eben so eine Photoshop-Alternative oder einen Browser. Aber man muss weder Baumeister noch Handwerker sein, um in der eigenen Wohnung einen Dübel zu setzen.

Ganz ähnlich verhält es sich beim Programmieren. Gut 20 Zeilen Python reichen zum Schreiben eines Passwort-Managers, der mehr als ein Spielzeug ist. Für den schnellen Einstieg ins Programmieren erklären wir anhand dieses Beispiels die wichtigsten Konzepte.

Ein Programm entwerfen

Ein Programm ist eine Anleitung, die der Computer von oben bis unten abarbeitet. Bevor Sie



Über das Web-basierte IPython Notebook können Sie die Programmiersprache Python ohne Installation ausprobieren.

die erste Zeile Code schreiben, sollten Sie eine klare Vorstellung haben, was der Computer machen soll. Programmieren fordert Sie heraus, große Probleme so klein aufzuteilen, dass der Rechner die Teilprobleme lösen kann. Mit zunehmender Erfahrung erarbeiten Sie immer mehr Lösungen für solche Teilprobleme.

Jedes gelöste Teilproblem kann im nächsten Programm wieder eingesetzt werden, um ein komplexeres Problem zu lösen. Wir empfehlen Ihnen, mit einfachen Beispielen anzufangen und sich erst nach und nach an komplexere Probleme zu wagen. Wenn Sie mit Python gelernt haben, große Probleme in lösbare Teilprobleme aufzubrechen, können Sie diese Erfahrung bei allen anderen Programmiersprachen und auch abseits vom Computer nutzen.

Für die Planung eines Programms empfehlen sich je nach Programm unterschiedliche Vorarbeiten. Für grafische Programme skizzieren Sie am besten zuerst die gewünschte Oberfläche. Für Datenbankanwendungen sollten Sie sich vorab überlegen, welche Daten das Programm speichert und unter welchen Bedingungen es darauf zugreift. Ein einfacher Passwort-Manager auf der Konsole muss nur die Eingabe und Verarbeitung der Daten in der richtigen Reihenfolge gewährleisten. Wir haben das Beispielprogramm c't SESAM genannt, was für „Sehr einfaches, sicheres Authentifizierungs-Management“ oder englisch „Super Easy Secure Authentication Management“ steht. Der Passwort-Manager erzeugt für jede Domain, bei der Sie einen Account haben, ein eigenes Kennwort. Damit Sie sich diese Passwörter nicht merken müssen, berechnet der Passwort-Manager diese, wenn Sie sie brauchen, aus dem Namen der Domain und einem Masterpasswort. Sie müssen sich also nur ein Masterpasswort merken und verwenden trotzdem auf jeder Website ein eigenes Passwort. Das Prinzip wurde in [1] genauer vorgestellt.

c't SESAM soll Anwender nach einem Masterpasswort und einer Domain fragen. Ohne diese beiden Werte kann das Programm nichts berechnen, also muss es diese Abfrage auf jeden Fall als Erstes ausführen. Anschließend verarbeitet der Passwort-Manager diese Daten so, dass auch ein Angreifer, der eines der generierten Passwörter erbeutet, daraus nicht das Masterpasswort und damit die Passwörter für die anderen Sites berechnen kann.

Dieses Teilproblem müssen Sie nicht selbst lösen, sondern können einen in Python integrierten Hash-Algorithmus (PBKDF2) benutzen. Kryptografische Algorithmen sollten Sie nie selbst implementieren, da es dort besonders viele Fallstricke und mögliche Fehler gibt.

Danach muss c't SESAM das Ergebnis in ein Passwort der richtigen Länge verwandeln. Für dieses Teilproblem wählt das Programm einzelne, für ein Passwort geeignete Zeichen, aus und hängt sie aneinander. Welche Zeichen das Programm auswählt, soll vom Ergebnis von PBKDF2 abhängen. Hat c't SESAM ein Passwort der gewünschten Länge zusammengebaut, gibt es das Kennwort zurück.

Python einrichten

Unter Linux gehört Python zur Standardausstattung. Achten Sie auf die Version: Die Anweisung `python` ruft meist Python in Version 2 auf. Der Befehl `python3` startet auf allen Distributionen die aktuelle Version der Sprache. Der hier vorgestellte Programmtext benötigt mindestens Python 3.2; aktuell ist 3.4.

Unter Windows müssen Sie Python erst installieren. Den Installer können Sie von python.org herunterladen. Auch hier sollten Sie die aktuelle Version von Python 3 installieren.

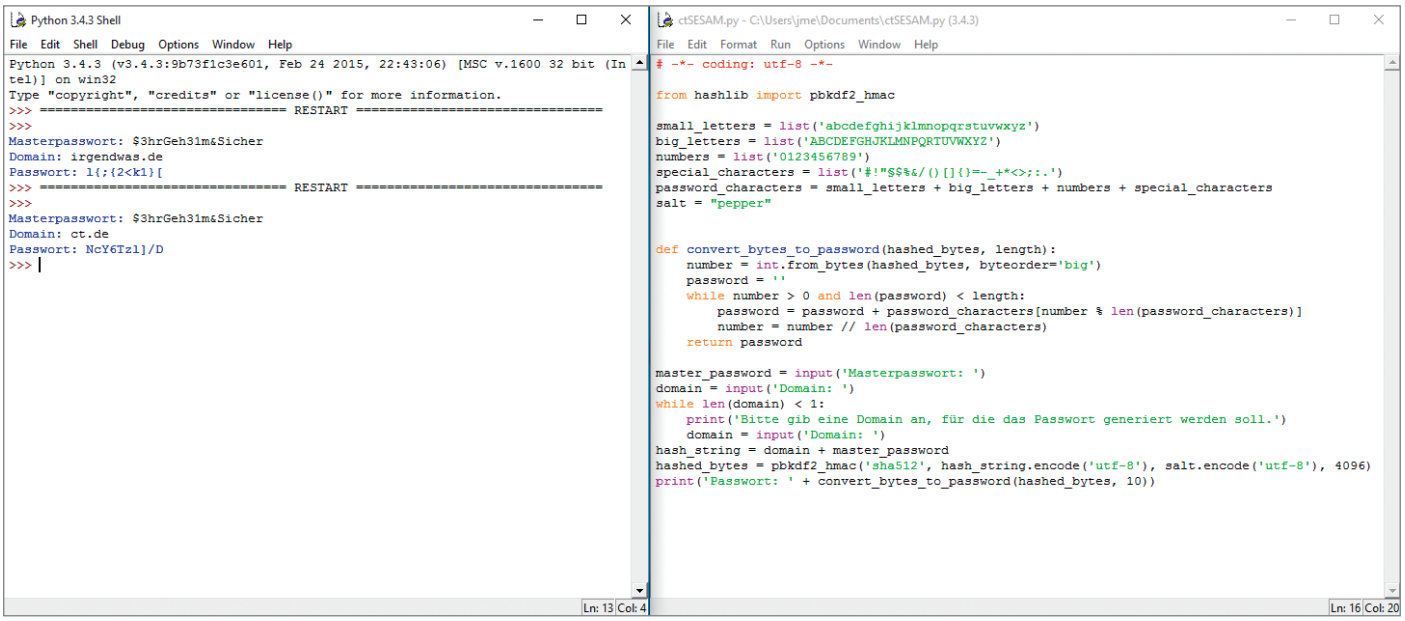
Wenn Sie kein Python installieren möchten, können Sie IPython Notebook im Browser verwenden. Dies ist ein Webdienst, mit dem man Python im Browser editieren und ausführen kann. Wenn Sie Mathematik-Programme kennen, werden Sie sich bei IPython Notebook sofort zu Hause fühlen. Jedes Notebook ist ein Dokument, das auf dem Server gespeichert wird. Es besteht aus Programmtext, den Ergebnissen der Ausführung und Text. Mit Maus oder Tastatur ausgewählte Abschnitte führen Sie mit dem Play-Symbol oben in der Menüleiste aus.

Im c't-Link finden Sie einen Dienst, der Ihnen virtuelle IPython-Notebook-Server zur Verfügung stellt. Echte Passwörter sollten über einen solchen Webdienst natürlich nicht generiert werden. Außerdem existieren die Testserver nur temporär, sodass die erzeugten Notebooks relativ schnell wieder gelöscht werden. Leider passiert das schon nach einigen Minuten Inaktivität, selbst wenn Sie das Browser-Fenster nicht schließen.

Eingaben speichern

Ein Python-Programm ist eine Textdatei mit der Endung `.py`. Unter Linux führen Sie es auf der Konsole mit `python3 ctSESAM.py` aus.

Unter Windows empfehlen wir eine Entwicklungsumgebung, aus der Sie das Pro-



Python für Windows enthält die Entwicklungsumgebung IDLE, die für c't SESAM ausreicht. Im rechten Fenster steht der Editor für das Programm, links die interaktive Konsole für die Ausgaben des Programms und kleine Tests.

gramm starten. IDLE (Integrated DeveLopment Environment) ist bei Python unter Windows dabei und reicht für c't SESAM aus. Die IDE startet in einer interaktiven Konsole, in der Sie Python-Befehle testen können. Ausgaben ihres Programms landen in diesem Fenster. Im Menü von IDLE legen Sie mit „File", „New File" eine neue Datei an und führen den Programmtext mit „Run", „Run Module" aus.

Damit das Python-Programm den Benutzer zu einer Eingabe auffordert, reicht der Befehl `input('Masterpassword: ')`. An den runden Klammern nach dem Namen erkennen Sie, dass `input()` eine Funktion ist. Funktionen sind Teile von Programmen, die eine bestimmte Aufgabe erfüllen. `input()` ist eine in Python integrierte Funktion, die Tastatureingaben entgegennimmt, bis die Eingabe-Taste gedrückt wird. Wie Sie eigene Funktionen definieren, lernen Sie etwas später bei der Konstruktion des Passworts. Funktionen wollen mit Parametern gefüttert werden:

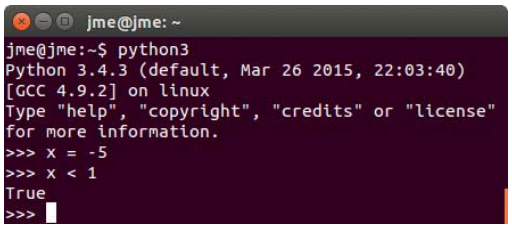
```
funktion(parameter1, parameter2)
```

Der Parameter `'Masterpassword: '` ist ein String, also eine Zeichenkette. Python akzeptiert Strings sowohl mit einfachen als auch mit doppelten Anführungszeichen. Wenn Sie das Programm ausführen, zeigt `input()` den in den Klammern übergebenen String an. So weiß der Benutzer, welche Eingabe das Programm erwartet.

Wenn Funktionen ein Ergebnis produzieren, geben sie es nach dem Aufruf zurück. Beispielsweise gibt `input()` die eingegebenen Zeichen als String zurück. Um den String für die spätere Verwendung zu speichern, müssen Sie eine Variable definieren. Variablenzuweisungen sehen aus wie mathematische Gleichungen, weisen den Rechner aber eigentlich nur an, in der Variablen links vom Gleichheitszeichen die Daten rechts vom = zu speichern. Eine Zeile reicht, um die eingegebenen Zeichen des `input()`-Befehls, der das Masterpasswort abfragt, in der Variablen `master_password` zu speichern:

```
master_password = input('Masterpassword: ')
```

Bei Python ist es üblich, Variablennamen, die aus mehreren Wörtern bestehen, durch Unterstriche zu verbinden. Falls Sie Wörter lieber direktAneinanderReihen wollen, akzeptiert Python das auch.



Was eine Anweisung zurückgibt, lässt sich auf der interaktiven Konsole ausprobieren. Gestartet wird sie auf der Konsole mit „python3" ohne Angabe eines Dateinamens oder unter Windows auch aus IDLE heraus (im Menü „Run"). Wer das IPython Notebook verwendet, muss nicht einmal eine Konsole starten, da die Anweisung dort direkt im Browser ausgeführt wird.

Fehlerhafte Eingaben

Die Zuweisung `domain = input('Domain: ')` bittet um die Eingabe der Domain und speichert sie in der Variablen `domain`. Gibt ein Benutzer keine Domain ein und drückt direkt die Eingabe-Taste, kann das Programm nicht weiter laufen, schließlich berechnet c't SESAM seine Passwörter anhand des Domain-Namens. Also sollte das Programm überprüfen, dass der Nutzer mindestens ein Zeichen eingegeben hat und ihn andernfalls erneut nach der Domain fragen.

Die Überprüfung einer Bedingung wird in Python mit `if Bedingung:` eingeleitet. Was passieren soll, wenn die Bedingung erfüllt ist, steht dann eingerückt in der nächsten Zeile. In Python werden zusammengehörende Programmabschnitte durch Einrückungen markiert. Es dürfen dafür Leerzeichen und Tabulatoren benutzt werden. Wichtig ist nur, dass ein Abschnitt einheitlich eingerückt wird.

Andere Programmiersprachen benutzen für die Markierung von zusammenhängenden Abschnitten meist geschweifte Klammern. Außerdem werden dort Befehle terminiert, häufig mit einem Strichpunkt, sodass mehrere Befehle in einer Zeile stehen können. Der für Python typisch verschwenderische Umgang mit Leerzeichen und Zeilensprüngen erzwingt lesbaren Programmtext, erfordert aber Disziplin beim Setzen von Leerzeichen.

Hinter dem `if` steht die Bedingung. Eine Bedingung ist eine Zeile Code, die entweder wahr (`True`) oder falsch (`False`) ist. Eine Variable vom Typ Boolean speichert genau einen dieser beiden Fälle. Im einfachsten Fall besteht eine Bedingung nur aus einer solchen Variablen. Häufig wird bei Bedingungen ein Vergleich benutzt wie `x < 1`: Wenn die Variable `x` eine Zahl kleiner als 1 ist, gibt der Vergleich `True` zurück.

Die Funktion `len()` gibt die Länge eines Strings als ganze Zahl zurück. Mit dieser Funktion kann c't SESAM prüfen, ob der Benutzer eine Domain eingegeben hat:

```
domain = input('Domain: ')
if len(domain) < 1:
```

```
    print('Bitte gib eine Domain an.')
    domain = input('Domain: ')
```

Die Funktion print() gibt seinen Parameter als eigene Zeile aus, ohne Eingaben zu erwarten.

c't SESAM stellt lediglich sicher, dass domain mindestens ein Zeichen lang ist, sodass der Benutzer auch etwas anderes als einen Domain-Namen eintippen kann. c't SESAM soll auch Eingaben verarbeiten, die nicht das Format von Internet-Domainnamen haben. Daher ist die Überprüfung bewusst minimalistisch. Wer statt „sparkasse-hannover.de" lieber „Bank" eintippt, kann den Passwort-Manager auch so nutzen.

Wenn ein Benutzer trotz des Hinweises den gleichen Fehler wiederholt, bliebe domain leer, obwohl die Überprüfung das eigentlich verhindern sollte. Wenn der Rechner etwas wiederholen soll, solange eine Bedingung erfüllt ist, hilft eine while-Schleife: Das Programm wiederholt den Programmtext in der Schleife, solange die Bedingung erfüllt bleibt. Um sicherzustellen, dass domain nicht leer bleibt, ersetzen Sie das if durch while. Die vier Zeilen zur Eingabe der Domain sehen dann so aus:

```
domain = input('Domain: ')
while len(domain) < 1:
    print('Bitte gib eine Domain an.')
    domain = input('Domain: ')
```

Daten vorbereiten

Der Algorithmus, den wir nicht selbst programmieren wollen (PBKDF2), erzeugt aus einer Folge von Bytes einen Schlüssel. Dazu muss das Programm die Domain und das Masterpasswort zu einem einzelnen String verbinden. In Python werden Strings mit + aneinandergehängt:

```
hash_string = domain + master_password
```

In der neuen Variablen hash_string stehen danach hintereinander die Domain und das Masterpasswort.

Der Aufruf von PBKDF2 erwartet vier Parameter: Den Anfang macht der Name des Hash-Algorithmus, den PBKDF2 intern verwendet. c't SESAM nutzt SHA512. Es folgt die Kombination aus Masterpasswort und Domain. Zusätzlich braucht PBKDF2 ein „Salz": Diese Bytefolge fließt in den ersten Hash-Durchgang des Algorithmus ein. Das Salz können Sie beliebig wählen, allerdings führt eine Änderung des Salzes zu komplett anderen Passwörtern. Wir haben „pepper" verwendet. Der letzte Parameter ist die Anzahl an Iterationen. Dieser Parameter bestimmt, wie schnell der Algorithmus läuft. Da Angreifer für jeden Versuch, Ihr Masterpasswort zu erraten, genauso lange brauchen wie Ihr Programm bei der Errechnung des Kennworts, sollte der Algorithmus nicht zu schnell laufen. Wir haben uns für 4096 Iterationen entschieden. [2] erklärt, wie PBKDF2 genau funktioniert.

Normalerweise kann man es der Programmiersprache überlassen, wie die Zeichen einer Zeichenkette als Byte-Folgen dargestellt werden. Da PBKDF2 aber bei Masterpasswort plus Domain und beim Salz statt Strings eine Folge von Bytes als Eingabe erwartet, müssen die Strings erst in Byte-Folgen konvertiert werden. string.encode() erledigt die Konvertierung. Damit die Funktion das richtige Encoding benutzt, müssen Sie den Namen des Encodings als Parameter übergeben:

```
hash_string_bytes = hash_string.encode('utf-8')
salt = "pepper"
salt_bytes = salt.encode('utf-8')
```

Das Encoding legt fest, welche Bytes im Speicher zu welchen Zeichen gehören. Gewöhnen Sie sich an, immer UTF-8 zu verwenden. Dieses Encoding umfasst alle Zeichen, was Probleme mit Sonderzeichen und Fremdsprachen vermeidet.

Debugging-Ausgaben

Auf der Suche nach Fehlern oder zur Überprüfung der Zwischenschritte werden Sie häufig wissen wollen, was in einer Variablen steht. Das erreichen Sie am einfachsten, indem Sie die Variable auf der Konsole anzeigen lassen.

Die print()-Funktion gibt Strings genau auf diese Weise aus und konvertiert automatisch andere Datentypen. print(5 / 2) schreibt zum Beispiel 2,5 auf die Konsole. Fügen Sie ruhig in Ihren Programmtext zusätzliche Zeilen mit print()-Anweisungen ein. Läuft das Programm erst einmal fehlerfrei, lassen sich diese Zeilen schnell wieder entfernen.

Bibliotheken nutzen

Machen Sie sich nicht selbst Arbeit, greifen Sie auf die Arbeit anderer Entwickler zurück! Python bietet Schnittstellen zu fast allen Bibliotheken. Eine Websuche nach dem Namen der Bibliothek und „Python" fördert in vielen Fällen auch eine nutzbare Dokumentation zutage. PBKDF2 ist Teil der hashlib, die zum Standardumfang von Python gehört und daher nicht nachinstalliert werden muss. Diese Funktion importieren Sie im Programm mit diesem Befehl:

```
from hashlib import pbkdf2_hmac
```

Danach lässt sich pbkdf2_hmac aufrufen, als sei die Funktion im Quelltext von c't SESAM definiert.

Nach der umfangreichen Vorbereitung der Daten ist der Aufruf des Algorithmus eher unspektakulär:

```
hashed_bytes = pbkdf2_hmac(
  'sha512',
  hash_string_bytes,
  salt_bytes,
  4096)
```

Es spricht übrigens nichts dagegen, Platz zu sparen und die Strings direkt bei der Übergabe an den Algorithmus in Bytefolgen zu konvertieren:

```
hashed_bytes = pbkdf2_hmac(
  'sha512',
  hash_string.encode('utf-8'),
  salt.encode('utf-8'),
  4096)
```

So entfällt die Definition von hash_string_bytes und salt_bytes. Das macht den Code zwar nicht schneller, aber kompakter.

Passwort erzeugen

PBKDF2 erzeugt eine Ausgabe mit 512 Bit, die c't SESAM in hashed_bytes als Folge von 64 Bytes speichert. Diese Bytes sind aber lediglich binäre Zahlen, die sich nicht einfach per Encoding in ein benutzbares Passwort verwandeln lassen.

Um ein Passwort zu erhalten, muss c't SESAM aus diesen Bytes einen String konstruieren, der aus Zeichen besteht, die sich für ein Passwort eignen. Da dies eine in sich abgeschlossene Aufgabe ist, empfiehlt es sich, diesen String in einer eigenen Funktion zu erzeugen.

Zu Beginn des Programms haben Sie schon die vordefinierten Funktionen input(), print() und len() aufgerufen. Jetzt lernen Sie eigene Funktionen zu schreiben. Mit Funktionen können Sie Programmtext, der mehrmals benötigt wird, zentral an einer Stelle definieren. Außerdem eignen sich Funktionen zur Strukturierung. Sie werden in Python mit dem Schlüsselwort def eingeleitet. Nach einer Leerstelle folgen der Name der Funktion und in runden Klammern ihre Parameter. Kommas trennen mehrere Parameter voneinander, keine Parameter sind auch erlaubt. Nach der schließenden Klammer folgt ein Doppelpunkt. In der nächsten Zeile beginnt der eigentliche Programmtext der Funktion, der wie bei der if-Anweisung beziehungsweise der while-Schleife eingerückt sein muss. Beispiel:

```
def addition(zahl1, zahl2):
  ergebnis = zahl1 + zahl2
  return ergebnis
```

Python kann den Aufruf einer Funktion erst verarbeiten, wenn sie vorher definiert oder importiert wurde. Daher müssen Sie den ganzen Block der Funktion zur Erzeugung des Passworts in Ihrem Programm oberhalb der ersten Programmzeile einfügen.

Die Funktion convert_bytes_to_password(), der komplexeste Teil von c't SESAM, soll jeweils ein Zeichen auswählen, das an das bisherige Passwort angehängt wird. Dafür muss für das Passwort zunächst die Variable definiert werden:

```
password = ""
```

c't SESAM verwendet für die Passwörter nur Zeichen, die üblicherweise von Websites angenommen werden. Wir haben das ganze Alphabet mit kleinen und großen Buchstaben ausgewählt, wobei wir zwei große Buchstaben ausgelassen haben, die leicht mit anderen Zeichen verwechselt werden. Das sind

das I, da es leicht mit dem l verwechselt wird, und das 0, da es der 0 ähnlich sieht. Zur Auswahl gehören zusätzlich die Ziffern 0 bis 9 und einige gebräuchliche Sonderzeichen. Sie können die Zeichenauswahl und Reihenfolge ändern, wenn Sie das möchten. Daraus ergäben sich aber Passwörter, die nicht mehr zu unserer Implementierung passen.

Damit convert_bytes_to_password() das Passwort Zeichen für Zeichen zusammensetzen kann, muss das Programm die Liste der Zeichen so vorbereiten, dass das leicht geht. Es bietet sich daher an, die Zeichen als Folge von Strings mit jeweils einem Zeichen vorzubereiten.

In Python verwendet man hierzu eine Liste. Diese wird in Python mit eckigen Klammern definiert ([element1, element2]) und kann verschiedene Datentypen enthalten. In anderen Programmiersprachen begegnet Ihnen diese Datenstruktur oft unter der Bezeichnung Array. Eine Möglichkeit wäre, die Liste als password_characters = ['a', 'b', 'c'] und so weiter zu definieren. Bei insgesamt 83 Zeichen würde das aber eine sehr mühsame Zeile.

Die Funktion list() konvertiert einen mehrere Zeichen langen String in eine Liste aus Strings mit je einem Zeichen. Damit lässt sich die Zeichenliste wesentlich platzsparender definieren. Zur besseren Übersicht haben wir die Buchstaben, Zahlen und Sonderzeichen als einzelne Listen definiert. Ähnlich wie Strings können diese Listen mit + aneinandergehängt werden:

```
lower_case_letters = list('abcdefghijklmnopqrstuvwxyz')
upper_case_letters = list('ABCDEFGHJKLMNPQRTUVWXYZ')
numbers = list('0123456789')
special_characters = list('#!"§$%&/()[]{}=-_+*<>;:.')
password_characters = lower_case_letters +↵
      upper_case_letters + numbers + special_characters
```

Insgesamt enthält die Zeichenliste password_characters damit alle 83 Zeichen, die im Passwort verwendet werden können. Um ein Zeichen aus der Liste auszuwählen, schreiben Sie einfach die Nummer des gewünschten Zeichens in eckigen Klammern hinter die Liste. Beispielsweise wählt password_characters[0] das a aus und password_characters[51] die 2.

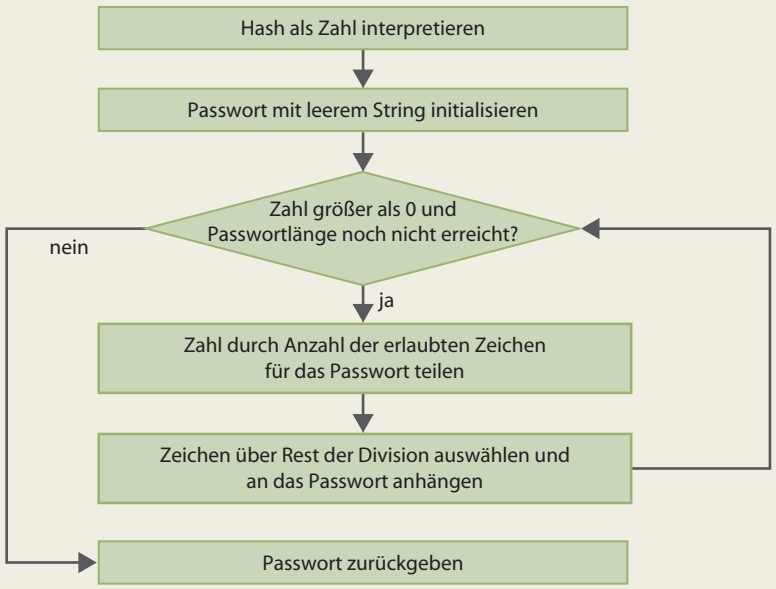
convert_bytes_to_password() soll das Ergebnis von PBKDF2 zur Auswahl der Zeichen verwenden. Dafür interpretiert das Programm die 64 Bytes als sehr große ganze Zahl.

```
number = int.from_bytes(hashed_bytes, byteorder='big')
```

Bevor Sie gelernt haben, mit Kommazahlen zu rechnen, haben Sie in der Schule bestimmt auch schriftlich dividiert. Dabei kam heraus, wie oft der Divisor in die Zahl passt und es blieb ein Rest übrig. c't SESAM nutzt diese Art der Division, um die Zeichen des Passworts auszuwählen: Teilt man number durch 83, bleibt ein Rest von 0 bis 82, der als Index für die Liste mit den Passwort-Zeichen dient.

Den Ganzzahlquotienten, also wie oft die Zeichenzahl in number passt, erhalten Sie mit dem //-Operator. Den sollten Sie nicht mit der normalen Division verwechseln, da zahl / andere_zahl in Python immer eine Gleitkommazahl ergibt, also eine Zahl mit Nachkommastellen.

Auswahl der Zeichen für das Passwort



Den Rest der Division berechnet der Modulo-Operator %. Mit number % len(password_characters) erhalten Sie also immer eine Zahl zwischen 0 und len(password_characters), der Anzahl der Zeichen in der Liste. In unserem Fall beträgt die 83. Mit dem Rest können Sie abhängig von number ein Zeichen für das Passwort auswählen und an das bisherige Passwort anhängen:

```
password = password + password_characters[
  number % len(password_characters)]
```

Damit c't SESAM nicht immer dasselbe Zeichen auswählt, muss das Programm anschließend die Ganzzahldivision ausführen, die number einen neuen Wert zuweist:

```
number = number // len(password_characters)
```

Im Prinzip können diese Schritte in einer while-Schleife so lange wiederholt werden, bis number 0 wird. Dadurch entstünde ein enorm langes Passwort. Das nimmt aber kaum eine Webseite entgegen. Deshalb übergeben Sie convert_bytes_to_password() die gewünschte Länge des Passworts als Parameter length. Um die Länge nicht zu überschreiten, hört c't SESAM auf, weitere Zeichen an das Passwort zu hängen, wenn len(password) < length unwahr wird. Die ganze Funktion sieht dann so aus:

```
def convert_bytes_to_password(hashed_bytes, length):
    number = int.from_bytes(hashed_bytes, byteorder='big')
    password = ''
    while number > 0 and len(password) < length:
        password = password + password_characters↵
                   [number % len(password_characters)]
        number = number // len(password_characters)
    return password
```

Ein logisches Und verknüpft die beiden Bedingungen in der while-Schleife. In der letzten Zeile gibt die Funktion das generierte Passwort zurück.

Ausgeben und beenden

Um die Funktion zur Passwortgenerierung zu benutzen, muss das Programm sie mit den passenden Parametern aufrufen. Der erste Parameter muss die Bytes enthalten, die PBKDF2 aus der Domain und dem Masterpasswort erzeugt hat. Als zweiter Parameter wird die gewünschte Länge des Passworts übergeben. Für die meisten Webseiten eignen sich zehn Zeichen. Wird ein längeres oder kürzeres Passwort benötigt, kann an dieser Stelle das Programm geändert werden. Wichtig: Diese Änderung verstellt die Erzeugung aller Passwörter. Sollten Sie schon mit c't SESAM erzeugte zehnstellige Passwörter einsetzen, müssen Sie den Wert wieder zurücksetzen, damit das Programm Ihnen diese Kennwörter wieder berechnen kann.

Das Ergebnis von convert_bytes_to_password() ist ein String, den Sie zusammen mit dem Hinweis, dass das Passwort folgt, über print() ausgeben können:

```
print('Passwort: ' +↵
      convert_bytes_to_password(hashed_bytes, 10))
```

Da nach dieser Zeile kein weiterer Programmtext folgt, beendet sich Python. Das Passwort bleibt auf der Konsole stehen, von wo aus Sie es abtippen oder kopieren können. Wenn Sie das Programm unter Windows mit einem Doppelklick starten möchten, können Sie hinter die letzte Zeile ein

input() setzen. Dann schließt sich das Fenster erst, nachdem Sie ein weiteres Mal die Eingabe-Taste gedrückt haben.

Umgang mit Fehlern

Wer programmiert, macht Fehler und sollte daher mit Fehlermeldungen umgehen können. Python reagiert bei Fehlern mit „Exceptions“, die eine ganze Menge darüber aussagen, was gerade falsch läuft. Es lohnt sich, absichtlich ein paar Fehler einzubauen, um auszuprobieren, wie die Programmiersprache reagiert. Bei einer Exception gibt Python standardmäßig ein Traceback aus. Darin steht ganz unten die Stelle, an der der Fehler wirklich aufgetreten ist. Darüber stehen alle Aufrufe, die an diesem Code beteiligt waren. Werden einer Funktion Parameter des falschen Typs übergeben, tritt der Fehler zwar innerhalb der Funktion auf. Die Fehlerquelle liegt aber in dem Aufruf, den das Traceback direkt darüber anzeigt. Vertauschen Sie mal beim Aufruf von convert_bytes_to_password die Parameter und führen Sie das Programm aus, um die Folgen zu beobachten.

Der Typ einer Exception sagt viel darüber aus, wo der Fehler liegen könnte. „SyntaxError“ deutet auf einen falsch formatierten Quelltext hin oder auf einen Tippfehler. Wenn Sie beispielsweise ein while in wile ändern, zeigt Python Ihnen sofort, in welcher Zeile der Fehler steht. Bei einem „TypeError“ wurde vermutlich eine Variable mit dem falschen Datentyp übergeben. Diesen Fehler produzieren Sie beispielsweise, wenn Sie die Parameter von convert_bytes_to_password vertauschen.

Die Meldung „TypeError: 'int' object is not iterable“ teilt Ihnen mit, dass int.from_bytes() einen Datentyp erwartet, der wie eine Liste aus einzelnen Elementen besteht. Im Traceback steht, in welcher Reihenfolge Sie die Parameter an die Funktion übergeben haben. IndexError deutet darauf hin, dass das Programm eine größere Liste erwartet hat. Schreiben Sie mal password_characters[85] hinter die Definition von password_characters und führen das Programm aus, um diesen Fehler zu provozieren.

Sollten Sie auf unerwartete oder unverständliche Fehler stoßen, hilft das Internet weiter. Wenn Sie den Text einer Fehlermeldung in eine Suchmaschine eingeben, fördert das in vielen Fällen Beiträge von Programmierern zutage, die das gleiche Problem hatten und deren Fragen schon beantwortet wurden. Suchen Sie dabei nach dem allgemeinen Typ des Fehlers, lassen Sie aber Eigenheiten des eigenen Programms wie den Traceback und Variablen- oder Funktionsnamen weg.

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
from hashlib import pbkdf2_hmac

lower_case_letters = list('abcdefghijklmnopqrstuvwxyz')
upper_case_letters = list('ABCDEFGHJKLMNPQRTUVWXYZ')
numbers = list('0123456789')
special_characters = list('#!"§$%&/()[]{}=-_+*<;:.')
password_characters = lower_case_letters + upper_case_letters + numbers + special_characters
salt = "pepper"

def convert_bytes_to_password(hashed_bytes, length):
    number = int.from_bytes(hashed_bytes, byteorder='big')
    password = ''
    while number > 0 and len(password) < length:
        password = password + password_characters[number % len(password_characters)]
        number = number // len(password_characters)
    return password

master_password = input('Masterpasswort: ')
domain = input('Domain: ')
while len(domain) < 1:
    print('Bitte gib eine Domain an, für die das Passwort generiert werden soll.')
    domain = input('Domain: ')
hash_string = domain + master_password
hashed_bytes = pbkdf2_hmac('sha512', hash_string.encode('utf-8'), salt.encode('utf-8'), 4096)
print('Password: ' + convert_bytes_to_password(hashed_bytes, 10))
```

Dies ist der gesamte Programmtext von c’t SESAM. Mit nur 22 Zeilen lassen sich in Python sichere, individuelle Passwörter für beliebige Domains berechnen.

c’t SESAM benutzen

c’t SESAM ist kein reines Anfängerbeispiel: Das Programm generiert sehr sichere Passwörter. Selbst wenn ein Angreifer das Salz kennt, müsste er einen hohen Aufwand treiben, um aus dem Passwort einer Domain das Masterpasswort zu berechnen. Auf einem i5 mit 3GHz dauert die Berechnung eines einzelnen Passworts gerade mal 0,04 s. Für das Knacken eines 10 Zeichen langen Masterpasswortes hingegen bräuchte der chinesische Supercomputer Tianhe-2, der derzeit schnellste Rechner der Welt, mit einer Brute-Force-Attacke schätzungsweise 5000 Jahre. Ist das Masterpasswort nur halb so lang, wäre er wahrscheinlich in weniger als einer Minute fertig. Denken Sie sich also ein Masterpasswort mit ausreichend vielen Zeichen aus.

Für die meisten Domains wird die Standardeinstellung eines 10 Zeichen langen Passworts mit Sonderzeichen ausreichen. Mit demselben Masterpasswort erzeugt c’t SESAM auf allen Rechnern dieselben Passwörter zu den gleichen Domains. Wenn Sie bei gleichem Domainnamen und Masterpasswort trotzdem andere Passwörter erzeugen wollen, ändern Sie das Salz auf einen anderen Wert. Auf diese Weise erzeugen Sie Ihr ganz persönliches SESAM. Wenn Sie Ihr SESAM auf verschiedenen Geräten einsetzen wollen, müssen Sie auf allen Geräten das gleiche Salz einstellen, damit dieselben Passwörter entstehen.

Für die Domain sollten Sie sich eine Methode überlegen, wie Sie von der URL auf die Domain kommen. Ob Sie „www.paypal.com“, „paypal.com“ oder lieber nur „paypal“ bei der Domain eintragen, hängt von Ihrer Vorliebe ab. Der Passwort-Manager generiert für jede Variante unterschiedliche Passwörter.

Manche Websites akzeptieren keine Sonderzeichen oder begrenzen die Länge des Passworts. In diesem Fall können Sie das Programm kurzzeitig abändern und die Sonderzeichen aus den password_characters ausklammern oder die Länge in der letzten Zeile anpassen. Sollten Sie für eine Domain ein neues Passwort brauchen, können Sie die Iterationszahl im Aufruf von pbkdf2_hmac um 1 erhöhen. Da alle vorherigen Passwörter mit den alten Einstellungen erzeugt wurden, sollten Sie das Programm nach diesem Sonderfall wieder auf die alten Einstellungen zurückändern. Außerdem sollten Sie sich notieren, welche Passwörter mit anderen Einstellungen erzeugt wurden, damit Sie immer das passende Passwort erzeugen. Da die Einstellungen einem Angreifer nur wenig nützen, können Sie Ihre Notizen auf einem Zettel notieren oder bei einem Cloud-Dienst ablegen.

(jme@ct.de)

Literatur

[1] Jürgen Schmidt, Eines für alle, Ein neues Konzept für den Umgang mit Passwörtern, c’t 18/14, S. 82

[2] Oliver Lau, Aus kurz wird lang, Passwörter sicher speichern mit Hilfe von PBKDF2, c’t 17/15, S. 180

c’t Quelltext, iPython-Notebook: ct.de/yvec

c’t SESAM überall

Wir werden diese Version von c’t SESAM in einer der nächsten Ausgaben um eine grafische Oberfläche mit Qt erweitern. Fans von C++ können sich auf eine Implementierung in dieser Sprache mit Qt-GUI und deutlich größerem Funktionsumfang freuen. Diese Version wird die Passworteinstellungen mit Ihrem privaten Server synchronisieren und auch Passwörter, die Sie nicht selbst ändern können, mit AES verschlüsselt speichern. Dazu kommen dann noch zwei Apps für Android: Ein c’t SESAM Passwort-Manager und eine App, die Ihre Passworteinstellungen über Ihren eigenen Server mit der Desktop-Version synchronisiert.