Unit 12a

Dr. Günter Kolousek

21. Juli 2015

Lege wiederum ein Verzeichnis an. Nennes es `21_unit12a`! In diesem Verzeichnis sollen alle Dateien der jeweiligen Einheit abgelegt werden.

1 Schulübungen

In diesen Beispielen wird die Verwendung

- des Datentyp Dictionary,
- die verschiedenen String-Methoden,
- einfache Eingabeprüfungen und
- der Aufbau eines einfachen Menüsystems

vertieft.

1. Die Cäsar-Chiffre ist eines der ältesten (und einfachsten) Verschlüsselungsverfahren. Ein Verschlüsselungsverfahren ermöglicht es, eine Nachricht mit Hilfe eines Schlüssels so zu kodieren, dass nur der Empfänger der Nachricht diese in Klartext lesen kann, soferne dieser ebenfalls im Besitz eines korrekten Schlüssels ist.

   Dazu funktioniert die Cäsar-Chiffre so, dass jedes Zeichen der Nachricht durch das Zeichen im Alphabet ersetzt wird, das sich um eine bestimmte Anzahl von Stellen (der Schlüssel!) weiter hinten im Alphabet befindet.

   Nehmen wir an, dass der Schlüssel 3 ist und wir als Zeichenvorrat lediglich das lateinische Alphabet heranziehen:

   ```
   A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
   | | | | | | | | | | | | | | | | | | | | | | | | |
   D E F G H I J K L M N O P Q R S T U V W X Y Z A B C
   ```

Mit Hilfe dieser Zuordnung kann man prinzipiell leicht die Nachricht (also den Klartext):

```
Maxi isst gerne Eis
```

verschlüsseln (chiffrieren), d.h. in den Geheimtext (oder auch Chiffre genannt) übersetzen. Allerdings stellt sich die Frage wie mit den Kleinbuchstaben und mit den Leerzeichen umzugehen ist.

Der Einfachheit halber werden wir die Kleinbuchstaben einfach vorerst in **Großbuchstaben wandeln** und die **Leerzeichen nicht verändern**!

Damit ergibt sich folgende Chiffre::

```
PDAL LVVW JHUQH HLV
```

Okay, jetzt an die Tasten! Schreibe eine Funktion `encrypt1(msg)` in einem Modul `caesar`, das eine Nachricht im Klartext als Parameter bekommt und den Geheimtext zurückliefert.

Zuerst wird der konzeptionell einfachste Weg bestritten. Wir sehen uns das Prinzip nochmals genau an und erkennen, dass jeder Buchstaben durch genau einen anderen Buchstaben ersetzt wird (so eine Chiffre nennt man Substitutionschiffre; substituieren = ersetzen).

Wie gehen wir das an? Hmm…, alle Zeichen durchgehen, jedes Zeichen in einen Großbuchstaben wandeln, danach durch den Geheimbuchstaben ersetzen und an den Ergebnisstring anfügen. Ja, so könnte das funktionieren.

Zwei kleine Probleme stellen sich uns:

a) Wie wandeln wir einen Buchstaben in einen Großbuchstaben um?

b) Wie führen wir die Ersetzung durch?

Zuerst zum ersten Problem: Welche Methoden haben die Strings in Python? Wie kann man das herausfinden? Wie immer entweder in der Hilfe nachsehen oder (die bevorzugte Lösung) direkt die Funktion `dir(s)` verwenden (Großbuchstaben … upper, Kleinbuchstaben … lower).

Nun zum zweiten Problem: Jedem Buchstaben ist ein anderer Buchstaben zuzuordnen. Aha, dafür kennen wir ja schon eine Datenstruktur (di..),

Nun ist alles restlos geklärt und es kann losgehen! Ausprobieren und testen!

2. Als nächstes ist jetzt die Funktion `decrypt1(msg)` zu schreiben, die einen Geheimtext bekommt und den Klartext zurückliefert.

   Wieder ein Problem! Wir können natürlich ein neues Dictionary anlegen, das uns zu einem Geheimtextbuchstaben den Klartextbuchstaben liefert, aber das ist keine besonders gute Lösung. Warum? Naja, wenn sich der Schlüssel ändert, dann müssen gleich zwei Dictionary geändert werden (ist eines zu ändern schon schlimm genug).

   Prinzipiell geht es ja gar nicht von einem "value" zu einem "key" zu gelangen, aber in unserem Fall geht das ja schon, da es zu jedem Buchstaben im Klartextalphabet (alle Zeichen aus denen ein Klartext bestehen kann) einen eindeutigen Buchstaben im Geheimtextalphabet gibt, der unterschiedlich zu allen anderen Buchstaben im Geheimtextalphabet.

   Schreibe zuerst eine Funktion `invert_dict(d)`, die genau das tut. Mit Hilfe dieser Funktion ist es ein Leichtes die Funktion `decrypt1(msg)` analog zur Funktion `encrypt1` zu schreiben. Tipp: Die Dicitionaries sollen der Einfachheit halber globale Variablen sein.

3. Soweit funktioniert die Verschlüsselung und die Entschlüsselung ja schon ganz gut, aber hat ein paar grundlegende Nachteile:

   - Der Schlüssel ist nicht leicht änderbar. Wenn jeder weiß, dass der Schlüssel immer 3 ist, dann ist es gar kein sicheres Verfahren (sicher ist die Caesar-Chiffre überhaupt nicht, davon aber später mehr).
   - Kleinbuchstaben werden immer zu Großbuchstaben.

   Also es muss ein anderer Ansatz her. Der erste Ansatz der Verwendung eines Dicitionary ist ja eigentlich gar nicht so schlecht, aber für dieses einfache Problem gar nicht notwendig, da das Alphabet ja immer um eine bestimmte Anzahl von Stellen verschoben werden.

   Damit man damit etwas anfangen kann, muss man wissen, dass die Zeichen im Computer nicht irgendwie auf Bits abgebildet sind, sondern durchaus eine Systematik aufweisen. Wie wir schon gelernt haben, wird jedem Zeichen intern ein Bitmuster zugeordnet. Diese Zuordnung nennt man eine Zeichencodierung. Dafür wird heutzutage meistens die Kodierung UTF-8 zur Speicherung und Übertragung von Texten verwendet (schaue nach was dein Texteditor bei den Einstellungen ausgewählt hat).

   Diese Kodierung hat eine interessante Eigenschaft, dass für die normalen lateinischen Buchstaben (ohne die Umlaute oder sonstige Sonderzeichen) jedem Zeichen genau ein Byte zugeordnet wird und dieses genau dem ASCII Standard entspricht. In dieser Zuordnung sind diese Buchstaben hintereinander angeordnet: Dem Buchstaben "A" ist der Ordnungswert 65, dem Buchstaben "B" der Wert 66 zugeordnet…

Jetzt muss man nur noch mehr den Wert 3 zu dem Ordnungswert des Klartextbuchstaben dazuaddieren und hat danach den Ordnungswert des Geheimtextbuchstaben.

Nächstes Problem wie kommt man zu diesem Ordnungswert? Und wie kommt man von solch einem Wert zu dem zugeordneten Buchstaben? Probiere die eingebauten Funktionen `ord(c)` und `chr(n)` aus! Ermittle auch die Werte für die Kleinbuchstaben. Kommen die in der Kodierung vorher oder nachher?

Gut, jetzt ist alles vorhanden für eine Neuimplementierung. Schreibe eine Funktion `encrypt2(msg, k)` und eine Funktion `decrypt2(msg, k)`. Der Einfachheit halber, sollen vorerst alle Zeichen wieder in **Großbuchstaben** gewandelt werden!

D.h. für jedes Zeichen ist der Ordnungswert zu bestimmen, der Key hinzu zuzählen, danach das Zeichen zu bestimmen und dieses zum Ergebnisstring hinten anzuhängen.

4. Funktioniert alles so wie gedacht? Ja? Nein? Liefert die Funktion `encrypt2("Maxi isst gerne Eis", 3)` genau den obigen Geheimtext?

   - Ja, gut dann mache beim nächsten Punkt weiter.
   - Nein, dann ist etwas falsch und du machst hier weiter.

   Bei genauer Betrachtung sehen wir, dass es sich eigentlich nicht um eine Verschiebung handelt sondern eigentlich um eine Rotation. Am Ende des Klartextalphabetes bewirkt eine Verschlüsselung, dass einem Zeichen, ein Zeichen am Anfang des Alphabetes zugeordnet wird, z.B. bei einem Key von 3:

   ```
   X Y Z
   | | |
   A B C
   ```

   `ord("X")` liefert 88, wenn man dann 3 hinzuzählt, dann kommt 91 heraus, aber `ord("A")` hat 65. Hmm, wenn man allerdings 26 abzieht, wenn der resultierende Ordnungswert größer als `ord("Z")` ist, dann könnte es doch funktionieren. Also: `91 > ord("Z")` also 91 - 26 = 65 und `chr(65) =` "A"=. Gut, dann das noch einbauen, damit:

   ```
   PDAL LVVW JHUQH HLV
   ```

   als Geheimtext herauskommt.

5. Als nächsten Schritt wollen wir auch die Kleinbuchstaben in die Verschlüsselung mit aufnehmen. Wir haben schon gesehen, dass die Großbuchstaben im ASCII oder UTF-8 Code von 65 bis 91 liegen und die Kleinbuchstaben liegen von 97 bis 122.

   Aber müssen wir das eigentlich wissen? Nein, da wir ja mittels `ord("a")` bzw. mittels `ord("z")` sowieso die Ordnungszahl bekommen. Wissen müssen wir lediglich,

dass die Buchstaben von a bis z bzw. von A bis Z jeweils hintereinander im Code angeordnet sind.

So, mit diesem Wissen könnten wir ja eigentlich schon die Kleinbuchstaben einbauen, wenn wir nur wüssten welcher Buchstabe ein Kleinbuchstabe und welcher ein Großbuchstabe ist. Aber auch dafür gibt es eine einfache Lösung in Form von Methoden des Typs `str`. Die Methodennamen sind so aufgebaut, dass man fragt: "ist dieser String ein Großbuchstabe?" und demzufolge liefert die Methode `True` oder `False` zurück.

Kopiere die Funktion `encrypt2` auf `encrypt3` und ändere diese entsprechend ab. Analoges Vorgehen zum Entschlüsseln!

Ist die Funktion korrekt implementiert, dann sollte folgendes Ergebnis zustande kommen:

```
>>> caesar.encrypt3("Maxi isst gerne Eis", 3)
'Pdal lvvw jhuqh Hlv'
```

6. Jetzt ist ein guter Zeitpunkt die Funktion `decrypt3(msg, k)` zu implementieren, sodass die Entschlüsselung von "Pdal lvvw jhuqh Hlv" korrekt funktioniert.

   Das ist aber ganz einfach! Was ist da zu der Funktion `encrypt3` lediglich unterschiedlich?

   Tipp: Einerseits muss der Key abgezogen anstatt addiert werden und andererseits muss die Bedingung angepasst werden.

7. So jetzt haben wir zwei Funktionen, die fast identisch sind. Können wir da nicht eine Funktion daraus machen, die mittels eines Parameters steuert, ob verschlüsselt oder entschlüsselt werden muss.

   Schreibe eine zusätzliche Funktion `translate_msg(mode, msg, k)`, die jetzt an erster Stelle einen Modus bekommt. Ist dieser Modus 1, dann soll verschlüsselt werden, ist dieser Modus -1, dann soll entschlüsselt werden. So einen Parameter oder Variable bezeichnet man als Flag (von engl. flag, also Fahne oder Flagge), die in einfach einen besonderen Zustand anzeigt.

   So eine Flagge kann z.B. entweder gehisst, auf halber Höhe heruntergelassen sein. Warum ist jedoch in diesem Fall gerade +1 und -1 gewählt worden?

8. Während man sich bei der Kryptographie damit beschäftigt sichere Chiffren zu entwickeln, versucht der Kryptoanalytiker Chiffren zu brechen, d.h. zu entschlüsseln ohne im Besitz des Schlüssels zu sein.

   Die einfachste Möglichkeit eine Chiffre zu brechen ist, alle verschiedenen Schlüssel auszuprobieren. Es gibt bei der Cäsar-Chiffre nur 26 verschiedene Schlüssel. Sie ist daher nicht sonderlich sicher.

Schreibe eine Funktion `brute_force(msg)`, die alle Schlüssel ausprobiert und die entschlüsselten Meldungen ausgibt.

Bei anderen Verschlüsselungsverfahren gibt es natürlich viel mehr verschiedene Schlüssel. So verwendet der Verschlüsselungsstandard AES Schlüssel, die 256 Bits lang sind. Wie viele verschiedene Schlüssel gibt es?

Tipp: Bei einer Dezimalzahl mit 3 Stellen gibt es insgesamt 1000 verschiedene Zahlen, bei einer Binärzahl mit 256 Stellen gibt es wieviele verschiedene Binärzahlen? Verwende Python um den exakten Wert zu bestimmen.

9. Jetzt geht es nur noch mehr darum, daraus noch ein kleines Programm zu machen:

   a) Schreibe dazu eine kleine Funktion `input_encrypt()`, die den Benutzer nach einer Nachricht und einem Schlüssel fragt, danach die Verschlüsselung durchführt und den Geheimtext ausgibt.

      Die Funktion soll sicherstellen, dass die eingegebene Nachricht nur aus alphabetischen Zeichen (Methode `isalpha`) oder aus Whitespace-Zeichen (Methode `isspace`) besteht.

      Whitespace-Zeichen sind…? Wenn die Funktion `translate_msg` derzeit nur auf das Zeichen `" "` abfragt, dann ist jetzt an der Zeit diese so abzuändern, dass diese ebenfalls die Methode `isspace` verwendet.

      Zur Abfrage des Schlüssels kann natürlich die normale Funktion `input` verwendet werden, allerdings kann dann "jeder" das eingegebene Passwort sehen. Verwende statt dessen die Funktion `getpass` aus dem Modul `getpass`!

      Weiters soll die Funktion sicherstellen, dass der Key immer eine ganze Zahl im Bereich von [1, 26] ist.

      Tätigt der Benutzer eine falsche Eingabe, dann muss er diese solange wiederholen bis diese korrekt ist.

      Zum Verschlüsseln soll die Funktion `translate_msg` verwendet werden.

   b) D.h. Das Modul `caesar` soll einerseits als Modul und andererseits als Programm funktionieren. Rufe die Funktion `input_encrypt` im Hauptprogramm auf.

   c) Schreibe weiters eine kleine Funktion `input_decrypt()`, die analog zu der Funktion `input_encrypt()` funktioniert.

   d) Schreibe eine Minifunktion `input_brute_force()`, die den Benutzer lediglich nach einer Nachricht fragt und danach die eigentliche `brute_force` Funktion aufruft. Die Nachricht ist analog auf Korrektheit zu überprüfen.

e) Entwickle jetzt eine Funktion `menu(title, actions)`, die ein (einfaches) Menü aufbaut und den Benutzer auswählen lässt.

Der Parameter `title` stellt den Titel dar, den der Benutzer sieht und daraufhin seine Wahl trifft.

Im Parameter `actions` befindet sich ein Dictionary, das als Keys die verschiedenen möglichen Eingaben enthält, die der Benutzer tätigen kann. Als Values sind die Funktionen enthalten, die bei der jeweiligen Eingabe durchgeführt werden soll.

Die Eingabe von ”q” oder ”quit” soll defaultmäßig zur Beendigung der Funktion führen. Alle anderen Eingaben bewirken nach Ausführung der entsprechenden Aktion wiederum eine Anzeige des Menüs. D.h. es handelt sich um eine Schleife, die erst dann beendet wird, wenn ”q” oder ”quit” eingegeben wird.

Der Aufruf der Funktion könnte folgendermaßen aussehen:

```
title = """Verschlüsseln einer Nachricht [e/encrypt]
Entschlüsseln einer Nachricht [d/decrypt]
Bruteforce [b/brute]
Beenden [q/quit]

Bitte wählen Sie! """

actions = {"e": input_encrypt,
           "encrypt": input_encrypt,
           "d": input_decrypt,
           "decrypt": input_decrypt,
           "b": input_brute_force,
           "brute": input_brute_force}

menu(title, actions)
```

f) Baue die Funktion `menu` in das Hauptprogramm ein, d.h. ersetze den Aufruf der Funktion `input_encrypt`.