Prof. Dr. Stefan Göller
Florian Bruse
Dr. Norbert Hundeshagen

Einführung in die Informatik

WS 2018/2019

Übungsblatt 2
1.11.2018-8.11.2018

Abgabe: Bis zum 8.11. 18:00 Uhr über moodle. Reichen Sie pro Aufgabe, die Sie bearbeitet haben, genau eine Textdatei mit dem Namen `aufgabe_i.py`, (wobei i die Aufgabennummer ist) ein, welche die Lösung ihrer Gruppe enthält.

Aufgabe 1 (ASCII, Funktionen, modulo) (10 Punkte):

Es soll ein Programm geschrieben werden, welches eine eingelesene Zeichenkette aus ASCII-Zeichen im Caesar-Code verschlüsselt.

Der Caesar-Code ist eine sehr einfache Verschlüsselungstechnik die auf der Verschiebung von Zeichen bzgl. eines Alphabets basiert. Um eine Zeichenkette zu verschlüsseln, benötigt man also ein Alphabet und eine positive Zahl, die die Verschiebung angibt. Die eingegebene Zeichenkette wird dann zeichenweise um die Position im Alphabet, die die Verschiebung vorgibt zyklisch nach rechts verschoben.

Beispiel: Nimmt man als Alphabet die 26 Kleinbuchstaben $a, b, c, \ldots, z$ und wählt die Verschiebung 2 ergibt sich also folgende Zuordnung der Buchstaben zu ihrer verschlüsselten Version.

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
c	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

Verschlüsselt man nun mit obiger Verschiebung die Zeichenkette `"zzgl"`, erhält man also `"bbin"`.

a) Schreiben Sie zunächst eine Funktion, die eine Zeichenkette als Eingabe erwartet und einen booleschen Wert ausgibt, der anzeigt ob diese Zeichenkette nur aus ASCII-Zeichen besteht.

   Hinweis: Zum Lösen der Aufgabe benötigen Sie die Funktion `ord()`.

   Machen Sie sich zunächst selber mit deren Funktionsweise[1] vertraut. Diese Funktion liefert die Unicode-Nummer eines eingebenen Zeichens zurück, wobei die ersten 0 bis 127 Nummern den ASCII-Zeichen entsprechen.

b) Wir betrachten nun als Alphabet die 128 ASCII-Zeichen. Schreiben Sie eine Funktion `caesar`, der eine Zeichenkette aus ASCII-Zeichen und eine positive natürliche Zahl übergeben werden können. Der Rückgabewert der Funktion soll die bzgl. des ASCII-Alphabets und der eingegebenen Zahl verschlüsselte Zeichenkette sein.

   Hinweis: Zum Lösen der Aufgabe benötigen Sie noch die Funktion `chr`, welche eine Zahl erwartet und das entsprechende Unicode-Zeichen zurück gibt. Dabei entprechen die ersten 127 Ziffern den ASCII-Zeichen. Bsp.: `chr(80)` gibt `"p"` zurück.

   Verschiebt man nun das Zeichen `"z"` um 3 Positionen erhält man das Zeichen `"}"` und um 16 Positionen das Zeichen `"\n"`. Beachten Sie, das letzteres ein sogenanntes Escape-Zeichen ist, welches vom `print`-Befehl als Zeilenumbruch interpretiert wird.

c) Schreiben Sie nun noch eine Funktion `un_caesar` zur Entschlüsselung einer Caesar-verschlüsselten Zeichenkette. Die Funktion erwartet Caesar-verschlüsselte Zeichenkette aus ASCII-Zeichen sowie die zur Verschlüsselung benutzte Verschiebung und gibt die entschlüsselte Zeichenkette zurück.

d) Benutzen Sie die obigen Funktionen, um nun ein Programm zu schreiben, welches zunächst abfragt ob eine Zeichenkette entschlüsselt oder verschlüsselt werden soll. Nach Eingabe der Zeichenkette und der Verschiebung soll dann die entsprechende entschlüsselte (oder verschlüsselte) Zeichenkette ausgegeben werden.

Aufgabe 2 (Funktionen, Dateien lesen/schreiben) (30 Punkte):

Es soll ein Programm geschrieben werden, überprüft, ob ein gegebenes Wort in einer Wortliste, gegeben durch eine Datei, enthalten ist. So etwas ist z.B. für Rechtschreibkorrektur nützlich. Als Basis dafür benutzen wir hier das unter der GPL-Lizenz verfügbare de_DE Hunspell-Wörterbuch, welches Sie in der Datei `de_DE_frami.dic` finden.

a) Schreiben Sie eine Funktion `find_pos` die eine Zeichenkette und einen Buchstaben als Parameter erwartet und die erste Position des Buchstaben in der Zeichenkette zurück gibt, falls dieser existiert und ansonsten `-1`.

[1] Zu finden unter https://docs.python.org/2/library/functions.html#ord

Beispiel: Bei Übergabe der Zeichenkette `"Abbild/JRTSm"` und des Buchstabens `"/"` soll die Funktion die Zahl `6` zurückgeben.

b) Schauen Sie sich zunächst den Inhalt der Datei `de_DE_frami.dic` an. Die Datei enthält neben einführenden Kommentarzeilen und einer Leerzeile (also einer, Zeile die nur einen Zeilenumbruch enthält) über 250000 Wörter der deutschen Sprache, wobei jedes Wort in einer eigenen Zeile eingetragen ist. Ferner werden Sie erkennen, dass den meisten Wortenden noch eine **Zeichenkette** beginnend mit `"/"` **angehängt** ist.

Erstellen Sie nun mit Hilfe obiger Funktion aus `de_DE_frami.dic` eine neue Datei `dictionary.txt`, welche nur noch zeilenweise die in `de_DE_frami.dic` vorkommenden Wörter ohne angehängte Zeichenkette enthält (also auch nicht die Kommentare und Leerzeilen).

Hinweis: Folgende Befehle benötigen Sie um Dateien zu manipulieren:

`f = open('test.txt','r')`	Öffnet eine Datei mit dem Namen `test.txt` im Lesemodus. Auf die Datei kann dann mit Hilfe des Bezeichners `f` zugegriffen werden.
`f = open('test.txt','w')`	Öffnet eine Datei mit dem Namen `test.txt` im Schreibmodus. Sollte die Datei nicht im Verzeichnis des Programms existieren, wird diese angelegt. Existierende Dateien werden überschrieben.
`f.close()`	Schließt die Datei die unter dem Bezeichner `f` geöffnet wurde. Dateien die nicht mehr benötigt werden, sollten aus Gründen des Speichermanagements immer geschlossen werden.
`f.read()`	Gibt den kompletten Inhalt der Datei als Zeichenkette zurück.
`f.readline()`	Gibt die erste Zeile der Datei als Zeichenkette zurück. Ein weiterer Aufruf von `f.readline()` gibt dann die nächste Zeile der Datei als Zeichenkette zurück, usw.. Sie erkennen das Ende der Datei daran, dass `f.readline()` eine leere Zeichenkette (also auch ohne Zeilenumbruch) zurückliefert.
`f.write("abc")`	Hängt die Zeichenkette `"abc"` an die letzte Zeile der Datei an, die unter dem Bezeichner `f` im Schreibmodus geöffnet wurde.

c) Schreiben Sie nun ein Programm, welches als Eingabe ein beliebiges Wort erwartet und die entsprechende Zeilennummer in `dictionary.txt` ausgibt, falls dieses Wort in der Datei enthalten ist.

Aufgabe 3 (Erweiterte Berechnung der Kubikwurzel) (10 Punkte):

Betrachten Sie noch einmal die näherungsweise Berechnung der Kubikwurzel einer Zahl mittels binärer Suche (Folie 183). Ihre Aufgabe ist es nun, das Verfahren so zu erweitern, dass es für alle Zahlen vom Typ `float` funktioniert, also auch Zahlen kleiner als 1.

Schreiben Sie dazu eine Funktion, die eine `float`-Zahl w und eine Fehlertoleranz $e > 0$ (auch als `float`) entgegen nimmt. Ihre Funktion soll nun prüfen, ob die Eingaben wie gewünscht sind. Danach soll ihre Funktion geeignete Startwerte für die Berechnung der Kubikwurzel mittels binärer Suche ermitteln (einer der Startwerte ist immer 0). Dann soll ihre Funktion, wie in der Vorlesung vorgestellt, einen Wert k für die Kubikwurzel suchen so dass k^3 nicht weiter als e von w entfernt ist. Dieser Wert soll dann ausgegeben werden.

Aufgabe 4 [Sternchenaufgabe] (Das Binärsystem):

Wir wollen Funktionen schreiben, welche Dezimalzahlen in Binärzahlen umwandeln und umgekehrt. Zur Wiederholung betrachten wir zunächst das vertraute Dezimalsystem. Die Zahl 1453 beispielsweise lässt sich wie folgt zerlegen:

$$1453 = 1 * 1000 + 4 * 100 + 5 * 10 + 3 * 1 = 1 * 10^3 + 4 * 10^2 + 4 + 10^1 + 3 * 10^0$$

Offensichtlich gilt also, dass der Wert der Zahl gebildet wird, indem man für jede Ziffer (ausgehend von rechts) eine entsprechende 10er-Potenz mit dem Wert der Ziffer selbst multipliziert, und dann alle dieser Werte aufsummiert. Beispielsweise bedeutet der Wert 3 der 0ten Ziffer von rechts, dass diese Summe 3 mal 10^0 enthalten soll. Dabei können diese Ziffern selbst Werte von 0 bis 9 annehmen. Für den Rest der Aufgabe schreiben wir Dezimalzahlen mit Subskript 10, also in der Form 1453_{10}.

Das Binärsystem funktioniert genauso, allerdings ist hier die Basis die 2. Das heisst, es kommen nur die Ziffern 0 und 1 vor, und es werden zum Summieren nicht 10er-Potenzen, sondern 2er-Potenzen verwendet. Wir kennzeichnen diese Zahlen mit Subskript 2. Die binäre Zahl 11011_2 hat also folgenden Wert:

$$11011_2 = 1*2^4+1*2^3+0*2^2+1*2^1+1*2^0 = 1*16_{10}+1*8_{10}+0*4_{10}+1*2_{10}+1*1_{10} = 27_{10}$$

a) Schreiben Sie eine Funktion, welche eine Binärzahl in eine Dezimalzahl umwandelt. Ihre Funktion soll also eine Zeichenkette entgegen nehmen, und dann überprüfen,

ob diese Zeichenkette eine gültige (positive) Binärzahl ist. Das heisst, dass diese Zeichenkette nur die Zeichen `0` und `1` enthält, und aussserdem nicht Länge 0 hat. Falls dies nicht der Fall ist, soll eine Fehlermeldung ausgegeben werden.

Ansonsten soll Ihre Funktion nun den Wert dieser Binärzahl als Zeichenkette zurück geben. Bei Eingabe `"11011"` soll der Rückgabewert also `"27"` sein (schauen Sie die Einführung an, um zu verstehen, wie dieser Wert ermittelt wird).

b) Schreiben Sie nun eine Funktion, welche eine (positive) Dezimalzahl in eine Binärzahl umwandelt. Ihre Funktion soll also wieder eine Zeichenkette entgegen nehmen, und prüfen ob sie eine Dezimalzahl kodiert. Dann soll die entsprechende Binärrepräsentation (als Zeichenkette) zurück gegeben werden. Bei Eingabe `27` soll Ihre Funktion beispielsweise `"11011"` zurück geben. Machen Sie sich bei der Berechnung folgende Sachverhalte zunutze:

   - Wenn Sie die Eingabe in einen `int` konvertieren, können Sie damit rechnen.
   - Die Binärrepresentation einer Dezimalzahl n hat höchstens k Ziffern, falls $n < 2^k$ (Spezialfall: 0).
   - Die $i$te Ziffer (beginnend von 0) der Binärrepräsentation von n ist 1 genau dann, wenn der Wert von `n // 2 ** i` eine ungerade Zahl ist. Alternativ können Sie die Ziffern der Binärrepräsentation folgendermaßen ermitteln: Ausgehend von der höchsten Ziffer schauen Sie, ob die entsprechende 2er-Potenz in die Zahl "passt", also ob Sie diese 2er-Potenz von der Zahl abziehen können, ohne eine negative Zahl zu erhalten. Wenn ja, ist die entsprechende Ziffer eine 1, ansonsten eine 0. Im ersten Fall machen Sie dann mit dem Rest weiter, um die restliche Binärdarstellung zu erhalten, im zweiten Fall mit der ursprünglichen Zahl.

     Beispielsweise ist die Binärdarstellung von 5_{10} die 101_2. Man erhält diese wie folgt: Aus dem vorigen Teil wissen wir, dass die Binärdarstellung 3 Ziffern hat. Wir testen nun für die linkeste Ziffer, ob die entsprechende 2er-Potenz (2^2) in die 5_{10} passt. Dies ist der Fall, denn $5_{10} - 2^2 = 1_{10} \geq 0$. Wir wissen also, dass die erste Ziffer eine 1 ist, und machen mit $1_{10} = 5_{10} - 2^2$ weiter. Für die zweite Stelle testen wir, ob 2^1 in 1_{10} passt - das ist nicht der Fall, denn $1_{10} - 2^1 = -1_{10} < 0$. Also ist die mittlere Ziffer eine 0, und wir machen mit 1_{10} weiter, um die letzte Ziffer zu erhalten. Dazu testen wir, ob 2^0 in 1_{10} passt. Das ist der Fall, also ist die letzte Ziffer eine 1, und die Binärdarstellung 101_2 von 5_{10} ist komplett.

Selbstverständlich lassen sich diese Aufgaben einfach durch den Aufruf von Bibliotheksfunktionen lösen. Versuchen sie es nur mit den Mitteln, die Sie in der Vorlesung kennen gelernt haben.