Prof. Dr. Stefan Göller
Daniel Kernberger

Einführung in die Informatik

WS 2019/2020

Übungsblatt 2
1.11.2019 - 7.11.2019

Abgabe: Bis zum 7.11.19 18:00 Uhr über moodle. Beachten Sie, dass ihre abgegeben Dateinamen genau so heißen wie verlangt.

Aufgabe 1 (Vokale ersetzen) (15 Punkte):

Schreiben Sie ein Python-Programm mit Dateinamen `aufgabe_1.py`, welches eine Zeichenkette einliest. Danach soll ihr Programm die eingelesene Zeichenkette in Kleinbuchstaben konvertieren mittels der Methode `lower()`. Wenn der Wert der Zeichenkette `s` also `"Hallo!"` ist, ist der Wert von `s.lower()` dann `"hallo!"`.

Danach soll ein Vokal eingegeben werden. Vokale sind `a, e, i, o` und `u`. Ihr Programm soll dann aus der vorigen, in Kleinbuchstaben umgewandelten Zeichenkette eine neue bilden, der sich von der vorigen dadurch unterscheidet, dass alle Vokale durch den eingebenen Vokal ersetzt werden.

Beispiel: Bei Eingabe `"Fischers Fritze fischt frische Fische"` und `"a"` soll Ihr Programm `"faschars fratza fascht frascha fascha"` ausgeben.

Aufgabe 2 (ASCII, Funktionen, modulo) (25 Punkte):

Es sollen zwei Programme geschrieben werden, eins das eine eingelesene Zeichenkette aus ASCII-Zeichen im Caesar-Code verschlüsselt und eins, das entschlüsselt.

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"`.

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

a) Wir betrachten nun als Alphabet die 128 ASCII-Zeichen. Schreiben Sie eine Programm `aufgabe_2_a.py`, das zuerst eine Zeichenkette aus ASCII-Zeichen und dann einen nichtnegativen Integer einliest, der die Verschiebung beschreibt. Es soll danach die bzgl. des ASCII-Alphabets und des Integers verschlüsselte Zeichenkette ausgegeben werden.

   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.

b) Schreiben Sie nun ein Programm `aufgabe_2_b.py` zur Entschlüsselung einer Caesar-verschlüsselten Zeichenkette. Das Programm liest zuerst die Caesar-verschlüsselte Zeichenkette aus ASCII-Zeichen ein und danach die zur Verschlüsselung benutzte Verschiebung (ein nichtnegativer Integer) und gibt die entschlüsselte Zeichenkette aus.

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

Aufgabe 3 (Das Binärsystem) (20 Punkte):

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:

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

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:

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

a) Schreiben Sie ein Programm `aufgabe_3_a.py`, welche eine Binärzahl in eine Dezimalzahl umwandelt. Ihr Programm soll also eine Zeichenkette entgegennehmen, und dann überprüfen, ob diese Zeichenkette eine gültige (positive) Binärzahl ist. Das heißt, dass diese Zeichenkette nur die Zeichen `0` und `1` enthält, und außerdem nicht Länge 0 hat. Falls dies nicht der Fall ist, soll eine Fehlermeldung ausgegeben werden.

   Ansonsten soll Ihr Programm nun den Wert dieser Binärzahl als Zeichenkette zurückgeben. Bei Eingabe `"11011"` soll der Rückgabewert also `"27"` sein (schauen Sie die Einführung an, um zu verstehen, wie dieser Wert ermittelt wird). Es ist ratsam, aber nicht notwendig den eingelesenen String in einen Integer zu konvertieren. Bitte vermeiden Sie führende Nullen bei der Ausgabe.

b) Schreiben Sie nun ein Programm `aufgabe_3_b.py`, welche eine (positive) Dezimalzahl in eine Binärzahl umwandelt. Ihr Programm soll also wieder eine Zeichenkette entgegennehmen, und prüfen ob sie eine Dezimalzahl kodiert. Dann soll die entsprechende Binärrepräsentation (als Zeichenkette) zurückgegeben werden. Bei Eingabe `27` soll Ihre Funktion beispielsweise `"11011"` zurückgeben. Machen Sie sich bei der Berechnung folgende Sachverhalte zunutze:

   - Wenn Sie die Eingabe in einen `int` konvertieren, können Sie damit rechnen.
   - Die Binärrepresentation einer Zahl n hat höchstens k Ziffern, falls $n < 2^k$ (Spezialfall: 0).

– Sie können 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.

Bitte vermeiden Sie bei Ihrer Ausgabe führende Nullen. Selbstverständlich lassen sich diese Aufgaben einfach durch den Aufruf von Bibliotheksfunktionen lösen. Versuchen sie es nur mit den Mitteln, die Sie in der Vorlesung kennengelernt haben.