Hochschule RheinMain
H. Arega, M.-A. Rohde, W. Weitz

Studiengang Medieninformatik
Wintersemester 2021/22

Programmieren 3

Prog3-Ueb-08

Aufgabe 1 (Funktionsargumente)

Abgleich von Argumenten. Definieren Sie bitte die folgenden Funktionen und werten Sie die folgenden Ausdrücke aus. Erklären Sie das jeweilige Ergebnis anhand des Argumentenabgleichs, wie in der Vorlesung vorgestellt:

```
def h(a,b,c=1000,*d,**e):
  print(a,b,c,d,e)

tup = ('ene','mene','mu' )
kw = { 'x':'iks', 'b':'beh', 'lst':[17,17,17] }
h(17,21), h(10,20,30), h(1,2,3,4,5,6,x=7, y=22)
h(1,2,3,4,5,6,c=7)
h(*tup), h(1,2,*tup,3), h(10, **kw)
h(10,20,*tup, **kw)
```

Aufgabe 2 (Funktionen, variable Argumentlisten, Textdatei lesen)

Schreiben Sie bitte eine **rekursive** Funktionen `ggTr`, die den größten gemeinsamen Teiler von zwei Zahlen berechnet. Erinnern Sie sich daran, dass

$$\forall x : \texttt{ggT}(x, x) = x$$

$$\forall x > y : \texttt{ggT}(x, y) = \texttt{ggT}(x - y, y)$$

$$\forall x, y : \texttt{ggT}(x, y) = \texttt{ggT}(y, x)$$

Testen Sie Ihre die Funktionen mit einigen Beispielen wie `ggTr(10,30)=10`, `ggTr(20,30)=10`, `ggTr(2,5)=1`, `ggTr(8,6)=2`, `ggtTr(7,3)=1`. Diese naive rekursive Version ist leider nicht sehr effizient (was nicht an der Rekursion, sondern an der Berechnungsweise hängt). Erinnern Sie sich, dass auch gilt

$$\forall x > y, x\%y \neq 0 : \texttt{ggT}(x, y) = \texttt{ggT}(x\%y, y) \tag{1}$$

Schreiben Sie bitte davon ausgehend eine **iterative** Funktion `ggT`, die den größten gemeinsamen Teiler von zwei Zahlen mit Hilfe von (1) berechnet.

Ihre Funktion "von Hand" mit vielen Werten zu testen ist langweilig. Also soll der Rechner das übernehmen. Lesen Sie bitte zeilenweise aus der Textdatei `ggtbeispiele.txt` (downloadbar aus read.MI). Jede Zeile enthält (durch Whitespace getrennt) drei Zahlen, wobei die letzte der ggT der ersten beiden ist. Berechnen Sie den `ggT` für alle Zeilen mit Hilfe Ihrer `ggT`-Funktion aus den ersten beiden Werten und überprüfen Sie, ob Ihr Ergebnis mit der "Musterlösung" in der jeweiligen dritten Spalte übereinstimmt. Bei Abweichungen soll die eingelesene Dateizeile und Ihre abweichende ggT-Lösung ausgegeben werden. Hinweis: Die Datei enthält drei fehlerhafte ggTs.

Schreiben Sie aufbauend auf Ihre `ggT`-Funktion eine Funktion `ggTl`, die *beliebig viele* Zahlen-Argumente akzeptiert und den ggT daraus berechnet.

Beachten Sie, dass der `ggT` dreier Zahlen `x,y` und `z` gleich dem `ggT` von dem `ggT` zweier Zahlen (zum Beispiel `ggT(x,y)`) und der dritten Zahl (zum Beispiel `ggT(ggT(x,y),z)`) ist.

Sie können also für `ggTl()` auf Ihre `ggT()`-Funktion zurückgreifen (einfach aufrufen – bitte kein Copy&Paste) und den ggT über alle übergebenen Zahlen schrittweise berechnen, indem Sie den ggT der ersten beiden Parameter berechnen und dann so lange zum Zwischenergebnis eine weitere Zahl dazu-ggT-en, bis alle Parameter abgearbeitet sind.

Die Funktion soll z.B. so nutzbar sein (Testwerte nach "=" angegeben):
`ggTl(10,80,20,75)=5` oder `ggTl(17,4)=1` oder `ggTl(7,2,2,1,20,11)=1`.

Aufgabe 3 (Textdatei lesen, Dictionaries, sortieren, argv)

Erschaffen Sie bitte ein Skript `zaehl.py`, um die Anzahl von Buchstaben, Wörtern und Textzeilen in Dateien zu ermitteln, ähnlich dem Unix-Shellkommando `wc`. "Wörter" sind dabei durch Whitespace (also Leerzeichen, Tabs, Zeilenumbrüchen usw.) getrennte Zeichengruppen ("Ich bin Bärenglocken-Verkäufer, woisch?" sind also *vier* Wörter). Wie bei `wc` soll der Name der Textdatei auf der Kommandozeile mitgegeben werden können. Der Name der einzulesenden Datei soll als Kommandozeilenparameter übergeben werden (also z.B. `python3 zaehl.py meintext.txt`).

Es soll zudem Funktionen `count_words` und `count_chars` geben, die jeweils ein Dictionary zurückgeben, in dem Wörter, bzw. Buchstaben auf Ihre Häufigkeit abgebildet werden, und zwar ohne Groß-/Kleinschreibung zu beachten (`and`, `And` und `AND` zählen also als drei Auftreten desselben Worts). Nutzen Sie Ihre Funktionen bitte nun, um die 25 häufigsten Wörter und Buchstaben einer Textdatei zu bestimmen und jeweils absteigend nach Häufigkeit sortiert auszugeben. Bitte testen Sie zunächst mit einer eigenen, kleinen Textdatei, wo Sie das Ergebnis gut überblicken können.

Wenn Sie mal eine größere Eingabe testen möchten: Was ist das 25-häufigste Wort in Shakespeare's "A Midsummer Night's dream"? Die zugehörige Textdatei finden Sie im read.MI. Dass die letzten zwei Zeilen in der Datei nicht zum Opus gehören, ignorieren wir mal (also einfach mitzählen).

Aufgabe 4 (Dateien, Datenstrukturen)

Im read.MI finden Sie eine Textdatei `fahrzeiten.txt`. Jede Zeile enthält vier durch Semikolon getrennte Felder, von denen das erste den Namen einer **Verkehrslinie** (Bus/Bahn), das zweite eine **Haltestelle**, das dritte die **nachfolgende** Haltestelle und das vierte die Zeit in **Minuten** zwischen den zwei Haltestellen enthält – allerdings leider völlig unsortiert. Im nachfolgenden Ausschnitt der Datei ist also z.B. angegeben, dass man mit der `S9` von Haltestelle `Flughafen` zum `Stadion` vier Minuten braucht.

```
S9;Stadion;Niederrad;3
Bus6;Welfenstrasse;Weidenbornstrasse;1
S9;Kelsterbach;Flughafen;6
Bus6;Ruhbergstrasse;Rothstrasse;1
S9;Flughafen;Stadion;4
U4;Festhalle/Messe;Bockenheimer Warte;2
```

Schreiben Sie bitte ein Python-Funktion `auskunft(linie, start, ziel)`, welche die genannte Datei einliest und für die angegebene Linie ein Tupel aus **Fahrzeit** und **Liste von Haltestellen** auf dem Weg von `start` nach `ziel` als Rückgabewert zurückgibt, wie unten im Beispiel gezeigt. Wenn `start` und `ziel` gleich sind, beträgt die Fahrzeit immer 0 Minuten, dazu gibt es keine Einträge in der Datei.

Sie können davon ausgehen, dass in den Fahrplandaten **je Linie** alle Stationen nur einmal als Starthaltestelle auftreten und die jeweils nachfolgende Haltestelle immer eindeutig ist (also eine einfache Sequenz von Haltestellen, nur "eine Richtung", keine Rückwege / Schleifen / Verzweigungen im Verlauf, kein "Umsteigen"), man muss sich also für die gewünschte Linie nur von Station zu Folgestation durchhangeln, bis das Ziel erreicht ist. Sie könne auch davon ausgehen, dass die angegebenen Haltestellen auf der übergebenen Linie existieren und nur tatsächlich von der Starthaltestelle erreichbare Ziele abgefragt werden.

Beispiel: Wie lange und über welche Stationen führt der Weg mit der Linie `S9` von `Kelsterbach` nach `Niederrad` (vgl. Beispieldaten oben):

- **Kelsterbach** nach Flughafen: 6 Minuten
- Flughafen nach Stadion: 4 Minuten
- Stadion nach **Niederrad**: 3 Minuten

Ergebnis: insgesamt 13 Minuten, Stationen: Kelsterbach,Flughafen,Stadion,Niederrad

Mit der zu implementierenden Funktion sieht das so aus:

```
>>> auskunft("Bus6", "Nordfriedhof", "Nordfriedhof")
(0, ['Nordfriedhof'])
>>> minuten,weg = auskunft("S9", "Kelsterbach", "Niederrad")
>>> print(minuten,"Minuten so:",weg)
13 Minuten so: ['Kelsterbach', 'Flughafen', 'Stadion', 'Niederrad']
```