Bitte bearbeiten Sie die Aufgaben in **Zweiergruppen**. Die Abgabe Ihrer Lösungen erfolgt bis spätestens Donnerstag der folgenden Woche **per E-Mail**

an: hans-georg.esser@hm.edu
Betreff: BS-2011, Übung 2, *Nachname1, Nachname2*

(Bitte beachten Sie diese Konvention, damit Ihre Mails nicht verloren gehen – bei der ersten Übung haben es nicht alle geschafft, den Betreff richtig zu setzen ☺)

6. Python: Dictionaries (3 Punkte)

Python kennt eine Datenstruktur namens „Dictionary“ (in anderen Programmiersprachen: assoziativer Speicher oder assoziatives Array). Ein leeres Dictionary `d` können Sie über

```
d = {}
```

erzeugen und dann Einträge in das Dictionary über

```
d[key] = value
```

vornehmen. Alternativ können Sie ein Dictionary bei der ersten Zuweisung mit Inhalt füllen; eine Kurzform von

```
passwords = {}
passwords["Anna"] = "geheim"
passwords["Bert"] = "secret"
passwords["Clara"] = "hidden"
```

wäre

```
passwords = {
  "Anna" : "geheim",
  "Bert" : "secret",
  "Clara" : "hidden" }
```

Schlüssel und Wert sind dabei also jeweils durch „:“ getrennt.

Den zu einem vorhandenen Schlüssel gehörenden Wert erfragen Sie genauso, wie Sie ihn zuweisen:

```
>>> print passwords["Anna"]
geheim
```

Geben Sie hier einen nicht vorhandenen Schlüssel an, erhalten Sie eine Fehlermeldung. Sie können vorab mit der Methode `has_key()` prüfen, ob es einen Schlüssel gibt:

```
>>> print passwords.has_key("Anna")
True
>>> print passwords.has_key("Anton")
False
```

Und schließlich können Sie einen Schlüssel auch mit dem Kommando `del` wieder entfernen:

```
>>> del passwords["Anna"]
>>> print passwords.has_key("Anna")
False
```

a) Das Programm `telefonbuch.py` (auf der Webseite, http://hm.hgesser.de/) implementiert ein sehr einfaches Telefonbuch mit Inverssuche (also Suche wahlweise nach dem Namen oder nach der Rufnummer) mit Hilfe von zwei Dictionaries (also mit redundanter doppelter Speicherung der Daten). Testen Sie das Programm, schauen Sie den Quellcode an und machen Sie sich mit der Funktionsweise vertraut.

Erklären Sie in Stichworten, wie die Suche funktioniert.

b) Erweitern Sie das Programm, so dass es auch einen Befehl `NEW` akzeptiert: Dann soll das Programm über `raw_input` zwei Variablen `name` und `nummer` einlesen und mit diesen Informationen beide Dictionaries aktualisieren. Wenn es den Namen oder die Rufnummer (oder beide) schon im Telefonbuch gibt, soll eine Fehlermeldung ausgegeben werden. Prüfen Sie, ob Ihr verändertes Programm funktioniert, indem Sie neue Einträge hinzufügen und sich anschließend die volle Liste ausgeben lassen.

c) Erweitern Sie das Programm, so dass es einen Befehl `UPDATE` akzeptiert. `UPDATE` soll wie `NEW` aus Aufgabe **b)** arbeiten, aber nur funktionieren, wenn es den Namen oder die Rufnummer schon gibt. Welcher der beiden Fälle vorliegt, muss das Programm selbst erkennen. Die Einträge in den beiden Dictionaries sollen dann entsprechend aktualisiert werden. Testen Sie die Änderungen.

d) Erweitern Sie das Programm, so dass es einen Befehl `DEL` akzeptiert. Danach soll es eine Variable `loeschen` einlesen und Einträge aus dem Dictionary löschen, falls `loeschen` eine der gespeicherten Rufnummern oder einer der gespeicherten Namen ist. Testen Sie die Änderungen.

e) Legen Sie eine Kopie `telefonbuch-1dict.py` von `telefonbuch.py` an und passen Sie das gesamte Programm so an, dass es auf die redundante Speicherung in zwei Dictionaries verzichtet; löschen Sie dazu die Definition des Dictionaries `lookup_number`.

7. Prozesse und Signale (2 Punkte)

Öffnen Sie ein Kommandozeilenfenster (Konsole, `xterm` etc.) und rufen Sie darin einen Editor auf, z. B. gedit:

```
gedit &
```

(Mit `&` starten Sie den Editor im Hintergrund und können in der Shell weiterarbeiten. Wenn gedit nicht installiert ist, müssen Sie ein anderes Programm starten, z. B. xeyes.)

Suchen Sie nun mit

```
ps auxw | grep gedit
```

nach dem Prozess und finden Sie seine Prozess-ID heraus. Mit dem Befehl `kill` können Sie dem Prozess Signale senden, z. B.

- `SIGSTOP` zum Anhalten des Prozesses (`kill -STOP` *`ID`*)
- `SIGCONT` zum Fortsetzen des Prozesses (`kill -CONT` *`ID`*)
- `SIGTERM` zum Beenden des Prozesses (`kill -TERM` *`ID`* oder `kill` *`ID`*)

Probieren Sie zunächst die ersten beiden Signale aus: Schicken Sie dem Editor das `STOP`-Signal und versuchen Sie dann, im Editor-Fenster Text einzugeben. Wechseln Sie danach zurück in die Konsole und schicken Sie dem Editor das `CONT`-Signal. Sehen Sie nun den Text, den Sie im gestoppten Zustand eingegeben haben?

Beenden Sie schließlich den Prozess, indem Sie ihm das `TERM`-Signal schicken.

a) Suchen Sie nach einem Prozess, der Ihnen nicht gehört. Was passiert, wenn Sie diesem ein Signal (z. B. SIGSTOP) schicken?

b) Lesen Sie die Manpage zu `killall` durch. Was würde passieren (nicht ausprobieren...), wenn Sie den Befehl `killall tcsh` bzw. `killall bash` (je nach Standardshell auf Ihrem Linux-System) eingeben? Falls Ihr Linux-System diese Manpage nicht enthält, finden Sie diese auch online: http://www.unix.com/man-page/1/killall/

8. Prozesszustände (1 Punkt)

Warum gibt es die Zustandsübergänge **a)** blockiert → laufend und **b)** bereit → blockiert nicht?