Unit 12b

Dr. Günter Kolousek

21. Juli 2015

Lege wiederum ein Verzeichnis an. Nennes es `22_unit12b`! In diesem Verzeichnis sollen alle Dateien der jeweiligen Einheit abgelegt werden.

1 Schulübungen

In diesen Beispielen wird die Verwendung

- des Datentyps Dictionary,
- von verschiedenen String-Methoden,
- von verschachtelte Schleifen mit verschachtelten Listen

vertieft bzw. gelernt.

1. Schreibe eine Funktion `input_sentences` in einem Modul `text_analyzer`, die den Benutzer nach beliebigen Sätzen fragt, diese in einer Liste abspeichert und danach zurückliefert. Das könnte also so aussehen:

   ```
   Bitte geben Sie die zu analysierenden Sätze ein (CTRL-D bricht ab):

   Satz: Maxi und Mini verliefen sich im Wald, aber bald.
   Satz: 10 alte Fledermäuse flogen im Wald
   Satz: Hat die alte Meisterhexe...
   Satz: Seid's gewesen. Denn als Geister...
   Satz: Erst hervor der alte Meister.
   Satz: <CTRL-D>
   ```

2. Füge dazu in diesem Modul einen Aufruf dieser Funktion ein, sodass das Modul sowohl als Modul als auch als Programm verwendet werden kann.

   Achtung: Das werden wir ab jetzt immer so machen und braucht nicht mehr explizit erwähnt werden!!

3. Schreibe eine Funktion `split_sentences(lst)`, die eine Liste solcher Sätze bekommt und eine Liste von Wortlisten zurückliefert, wobei die übergebene Liste `lst` direkt verändert werden soll.

   Die Wortlisten entstehen indem man die Sätze an den Whitespace Zeichen und den Satzzeichen (`"',;.:-`) trennt. Die Satzzeichen werden nicht benötigt.

   Gehe dazu folgendermaßen vor:

   a) Ersetze zuerst alle Satzzeichen hintereinander durch Leerzeichen. Verwende dazu die `replace` Methode des Typs `str`.

   b) Trenne (splitte) danach den String in die einzelnen Wörter mittels der Methode `split` des Typs `str`.

   Für die obigen Daten sieht das Ergebnis dieses Funktionsaufrufes folgendermaßen aus:

```
>>> lst = ['Maxi und Mini verliefen sich im Wald, aber bald.',
... '10 alte Fledermäuse flogen im Wald',
... 'Hat die alte Meisterhexe...',
... "Seid's gewesen. Denn als Geister...",
... 'Erst hervor der alte Meister.']
>>> split_sentences(lst)
[['Maxi', 'und', 'Mini', 'verliefen', 'sich', 'im', 'Wald',\
'aber', 'bald'], ['10', 'alte', 'Fledermäuse', 'flogen', 'im', 'Wald'],\
['Hat', 'die', 'alte', 'Meisterhexe'], ['Seid', 's', 'gewesen', 'Denn',\
'als', 'Geister'], ['Erst', 'hervor', 'der', 'alte', 'Meister']]
>>> lst
[['Maxi', 'und', 'Mini', 'verliefen', 'sich', 'im', 'Wald',\
'aber', 'bald'], ['10', 'alte', 'Fledermäuse', 'flogen', 'im', 'Wald'],\
['Hat', 'die', 'alte', 'Meisterhexe'], ['Seid', 's', 'gewesen', 'Denn',\
'als', 'Geister'], ['Erst', 'hervor', 'der', 'alte', 'Meister']]
```

   Bei der obigen Ausgabe wurden zur besseren Lesbarkeit die Zeilen manuell umgebrochen und der Zeilenumbruch jeweils mit einem \ gekennzeichnet.

   Baue diese Funktion in das Hauptprogramm ein.

4. Schreibe eine Funktion `purge_bad_words(lst)`, die eine Liste von Listen von Wörtern erhält und alle diejenigen herauslöscht, die

   - weniger als 2 Zeichen lang sind oder
   - nicht alphabetische Zeichen enthalten

   Tipps:

- Um herauszufinden, ob ein String nur aus alphabetischen Zeichen besteht, kann die Methode `isalpha` verwendet werden.
- Löschen kann man mit der `del` Anweisung: `del lst[3]` löscht das Element mit dem Index 3 aus der Liste.

Die übergebene Liste `lst` soll direkt manipuliert und auch zurückgeliefert werden::

```
>>> purge_bad_words(lst)
[['Maxi', 'und', 'Mini', 'verliefen', 'sich', 'im', 'Wald', 'aber',\
'bald'], ['alte', 'Fledermäuse', 'flogen', 'im', 'Wald'], ['Hat',\
'die', 'alte', 'Meisterhexe'], ['Seid', 'gewesen', 'Denn', 'als',\
'Geister'], ['Erst', 'hervor', 'der', 'alte', 'Meister']]
```

Wie leicht zu sehen sind sind die zu kurzen Wörter und die Wörter, die nicht-alphabetische Zeichen enthalten entfernt worden.

Tipp: Wenn über eine veränderbare Collection iteriert wird, dann gibt es Probleme einzelne Elemente aus dieser Sequenz zu löschen. Prinzipiell gibt es mehrere Ansätze wie mit dieser Situation umgegangen werden kann.

Ohne weiteres Spezialwissen geht es mit einer `while`Schleife, die von oben nach unten zählt. Aber es funktioniert auch mit einer `for`-Zählschleife, die von oben nach unten zählt:

```
for i in range(len(sentence) - 1, -1, -1):
```

`sentence` gibt hier die Liste von Wörter an, die einen einzelnen Satz ausmachen.

Der dritte Parameter von `range` gibt die Schrittweite an, die defaultmäßig immer als 1 genommen wird. Da wir in diesem Fall jedoch hinunterzählen wollen, wird hier die Schrittweite als -1 angegeben.

Will man eine iterator-basierte Schleife verwenden, dann benötigt man eine Kopie der ursprünglichen Liste. So eine Kopie kann man entweder manuell erstellen (mühsam) oder mittels desm Moduls `copy` und der darin enthaltenen Funktion `copy`:

```
import copy

lst = [1,2,3]
lst_copy = copy.copy(lst)
```

Achtung beim Zählen des Index!

5. Schreibe eine Funktion `analyze_words(lst)`, die eine Liste wie aus dem vorhergehenden Punkt als Argument bekommt und ein Dictionary mit allen Worten als Keys und deren Häufigkeiten als Values zurückliefert, wobei aber alle Wörter nur klein geschrieben gezählt werden::

```
>>> analyze_words(lst)
{'wald': 2, 'mini': 1, 'aber': 1, 'gewesen': 1, 'flogen': 1, 'im': 2,\
'hervor': 1, 'als': 1, 'verliefen': 1, 'maxi': 1, 'sich': 1,\
'meister': 1, 'hat': 1, 'bald': 1, 'meisterhexe': 1, 'erst': 1,\
'fledermäuse': 1, 'geister': 1, 'die': 1, 'alte': 3, 'und': 1,\
'denn': 1, 'seid': 1, 'der': 1}
```

6. Schreibe eine Funktion `analyze_letters(lst)`, die wiederum eine Liste der Liste der Wörter bekommt und ein Dictionary mit Buchstaben als Keys und deren Häufigkeiten als Values zurückliefert. Wiederum werden alle Buchstaben als Kleinbuchstaben betrachtet:

   ```
   >>> analyze_letters(lst)
   {'a': 10, 'c': 1, 'b': 2, 'e': 28, 'd': 9, 'g': 3, 'f': 3, 'i': 12,\
   'h': 4, 'm': 7, 'l': 10, 'o': 2, 'n': 7, 's': 9, 'r': 10, 'u': 2,\
   't': 8, 'w': 3, 'v': 2, 'x': 2, 'ä': 1}
   ```

7. Schreibe nun eine Funktion `purge_analyzed_letters(dic)`, die alle Umlaute (ä, ü, ö) aus dem Dictionary (des vorhergehenden Punktes) entfernt::

   ```
   >>> purge_analyzed_letters(dic)
   {'a': 10, 'c': 1, 'b': 2, 'e': 28, 'd': 9, 'g': 3, 'f': 3, 'i': 12,\
   'h': 4, 'm': 7, 'l': 10, 'o': 2, 'n': 7, 's': 9, 'r': 10, 'u': 2,\
   't': 8, 'w': 3, 'v': 2, 'x': 2}
   ```

   Tipp: Auch hier kann die `del` Anweisung verwendet werden. Der Einsatz ist analog wie bei Listen, nur wird anstatt dem Index der Key verwendet.

8. Entwickle eine Funktion `sort_letters(dic)`, die ein Dictionary wie aus dem vorhergehenden Punkt bekommt und eine absteigend sortierte Liste von Tupel mit Buchstabe und Häufigkeit zurückliefert:

   ```
   >>> sort_letters(dic)
   [('e', 28), ('i', 12), ('a', 10), ('l', 10), ('r', 10), ('d', 9),\
   ('s', 9), ('t', 8), ('m', 7), ('n', 7), ('h', 4), ('g', 3), ('f', 3),\
   ('w', 3), ('b', 2), ('o', 2), ('u', 2), ('v', 2), ('x', 2), ('c', 1)]
   ```

   Information:

   Was kann man hier erkennen? Dass der Buchstabe ”e” der am häufigsten verwendete Buchstabe in der deutschen Sprache ist. Genau diesen Angriff haben die Kryptoanalytiker auf monoalphabetische Substitutionsverfahren – also Verschlüsselungsverfahren wie die Cäsar-Chiffre eine ist – durchgeführt.

   Wenn man nämlich in einem Geheimtext den häufigsten Buchstaben sucht und annimmt, dass dieser ein ”e” ist, hat man einen ersten Anhaltspunkt. Der zweithäufigste Buchstabe im deutschen Alphabet ist das ”n”, das bei unserem kurzem

Text aber relativ weit hinten kommt. Danach kommt in der deutschen Sprache schon das ”i”, das auch in unserem kurzem Text an nächster Stelle kommt. Danach folgen in der ”High-Score-Liste” die Buchstaben ”r”, ”s”, ”t” und ”a” weitgehend gleich häufig…

Wie haben die alten Kryptographen im 16.Jh. versucht das Problem zu lösen? Nicht ein Alphabet nehmen, sondern mehrere Alphabete! Die Vigenère-Chiffre war die Lösung gegen einen derartigen Angriff, aber davon später einmal mehr.

9. Probiere aus:

   a) `"abcdef".index("c")`

   b) `"abcdef".index("g")`

   c) `"abcdef".index("de")`

   d) `"c" in "abcdef"`

   e) `"g" in "abcdef"`

   f) `"de" in "abcdef"`

   g) `"abcdef"[2:4]`

   h) `"abcdef"[2:]`

   i) `"abcdef"[-1]`

   j) `"abcdef"[2:-1]`

   k) `[1, 2, 3, 4, 5].index(3)`

   l) `[1, 2, 3, 4, 5].index(9)`

   m) `[1, 2, 3, 4, 5][5]`

   n) `[1, 2, 3, 4, 5][2:4]`

   o) `[1, 2, 3, 4, 5][-2]`

   p) `[1, 2, 3, 4, 5][5:9]`

   q) `[1, 2, 3, 4, 5][4:]`

   r) `len("abc") + len(range(3)) + len({1, 2, 3}) + len({1: 2, 2: 3})`

   s) `"abc".startswith("a")`

   t) `"abc".startswith("b")`

   u) `"abc".startswith("ab")`

   v) `"abc".endswith("bc")`

   w) `print("X" + " ab c ".strip() + "X"`

x) `",".join(["a", "b", "c"])`