Und nun kann wie gehabt aus `d` die Liste der Anagrammgruppen ermittelt werden.

Oder wir wählen eine alternative Syntax, so genannte *list-comprehensions*. Diese lehnen sich etwas an die aus der Mathematik bekannte Mengennotation an. Sie sind nützlich, wenn aus gegebenen Listen (oder anderen Sequenzen) neue erzeugt werden sollen. **Beispiele**:

```
>> [x**2 for x in range(5)]
[0, 1, 4, 9, 16]
>> [x for x in range(10) if x**2%5==1]
[1, 4, 6, 9]
```

Das können wir auch verwenden um eine Liste von Anagrammgruppen zu erzeugen:

```
[ag for ag in d.values() if len(ag) > 1]
```

Wir entschließen uns nun noch auch die Gesamtzahl der Wörter in den Gruppen von Anagrammen mit der (seit Version 2.3) eingebauten Funktion `sum()` zu berechnen, der wir das Ergebnis einer *list-comprehension* als Agument übergeben, die Funktion `clock()` aus dem Modul `time` zu importieren, damit wir auf der Computeruhr nachsehen können wie spät es ist – und daraus die Laufzeit unseres Programms zu berechnen.

Damit gelangen wir zu folgendem Code:

8. Das fertige Programm `anagramme.py`

```
from time import clock
def anagramm_signatur(wort):
    buchstaben = list(wort.lower())
    buchstaben.sort()
    return ''.join(buchstaben)

def finde_anagramme(woerter):
    d = {}
    for wort in woerter:
        d.setdefault(anagramm_signatur(wort), []).append(wort)
    return [ag for ag in d.values() if len(ag) > 1]

t1 = clock()
woerter = file("wordlist.txt").read().splitlines()
anagruppen = finde_anagramme(woerter)
t2 = clock()
print "Berechnungszeit: %5.2f s." % (t2-t1)
print "Es gibt %d Anagrammgruppen" % len(anagruppen)
print "mit insgesamt %d Wörtern" % sum([len(ag) for ag in anagruppen])
print
raw_input("Eingabe-Taste druecken\n")
for anagramme in anagruppen:
    print '\t'.join(anagramme)
```

Ein Programmlauf liefert mir (auf einem 2.66 GHz- Rechner unter Windows-XP) das folgende Ergebnis:

```
Berechnungszeit:  0.53 s.
Es gibt 2531 Anagrammgruppen
mit insgesamt 5683 Wörtern
Eingabe-Taste druecken
Remus       serum
horse       shore
strain      trains
disowned    downside
bluer       ruble
Akron       Koran
fierce      Recife
Erich       Reich
bluest      bustle      subtle
....
```

9. Kurze Schlussbetrachtung

Sehen wir einmal von der Ausgabe ab, so finden wir, dass knapp mehr als zehn Zeilen Code reichen um die Anagrammgruppen aus `wordlist.txt` zu berechnen. Der Code hat eine klare Struktur und ist meines Erachtens gut lesbar.

Obwohl Python eine interpretierte Sprache ist, hat das Programm ein äußerst praktikables Laufzeitverhalten, das noch dazu proportional zur Anzahl der zu verarbeitenden Wörter in der Textdatei ist. Dies liegt natürlich daran, dass so leistungsfähige Datentypen wie Listen und *dictionaries* in Python eingebaut und hochgradig optimiert sind.

Anregung

Ich wäre sehr interessiert an äquivalenten anagramm-Programmen, die in anderen Programmiersprachen (C (C++), Pascal (Delphi), Java, VisualBasic, PHP, Perl, usw.) implementiert sind (oder vielleicht sogar in dem als Wollmilchsau berühmten Excel + VBA), um zu sehen:

- welche Unterschiede gibt es im Programmieraufwand in verschiedenen Sprachen
- welche Unterschiede gibt es im Laufzeitverhalten

Sollten Sie, geneigter Leser, solche zufällig haben – oder erzeugen wollen, senden Sie mir bitte ein Exemplar davon an *glingl@aon.at* . Wenn Sie Kommentare oder Vorschläge zu der hier gezeigten Problemlösung haben, sind diese natürlich auch sehr willkommen.

Sollten ein paar davon zusammenkommen, könnte vielleicht eine kleine Zusammenschau für dieses Blatt dabei herausschauen.

In dieser Folge haben wir schon stark von den Fähigkeiten von Python-Objekten Gebrauch gemacht, deren Typ in Python eingebaut ist. In der nächsten Folge soll es um objektorientierte Programmierung im engeren Sinn gehen: um die Programmierung benutzerdefinierter Klassen.

Literatur

Zusätzlich zu den in PCNEWS-84 genannten Quellen ist inzwischen erschienen:

Michael Weigend: Python GE-PACKT in der GE-PACKT-Reihe des mitp-Verlags. Ein sehr preisgünstiges und nützliches Nachschlagewerk das aber auch viele kurze und klare Beispiele enthält.