Übungsblatt 5

Willkommen zum Praktikum zu Programmieren in Python.

Aufgabe 1. Schreiben Sie ein Modul `wc`, das Ihnen Funktionen zur Extraktion von Informationen über Dateien zur Verfügung stellt. Alle Funktionen haben als ersten Parameter einen String als Dateinamen. Die Funktion `chars` gibt die Anzahl der Zeichen, die Funktion `words` die Anzahl der Wörter und die Funktion `lines` die Anzahl der Zeilen zurück. Es soll weiterhin eine Funktion `wc` geben, die alle oben genannten Informationen zurück gibt und eine Funktion `wc_show`, die diese Informationen ausgibt, zum Beispiel:

```
Die Datei /usr/share/dict/words hat
   99171 Zeilen
   99171 Wörter
  938587 Buchstaben
```

Die Ausgabe der Funktion `wc_show` kann in anderen Sprachen erfolgen. Die Sprache wird mit der Funktion `set_lang` eingestellt und akzeptiert die Parameter `'de'` für Deutsch und `'en'` für Englisch. Es gibt ein Kommandozeilenprogramm `wc`, das dieselben Werte ausgeben sollte (bei Buchstaben die Anzahl der Bytes).

Aufgabe 2. Schreiben Sie ein Modul `count`, das die Vorkommen von Buchstaben und Wörtern in Dateien zählt. Es soll die Funktion `count_words` und `count_chars` geben, die jeweils ein Wörterbuch zurückgeben in dem die Wörter beziehungsweise Buchstaben auf Ihre Häufigkeit abgebildet werden.

Fassen Sie das Modul `count` und das Modul `wc` in einem Modul `text` als Package zusammen und stellen Sie sicher, dass beide Untermodule beim Laden des Moduls schon zur Verfügung stehen.

Nutzen Sie das Modul `text` in einem Programm `haeufig.py`, um die 25 häufigsten Wörter und Buchstaben zu bestimmen und auszugeben. Was ist das 25-häufigste Wort in Shakespeare's "A Midsummer Night's dream"?

Aufgabe 3. Implementieren Sie den abstrakten Datentyp Menge als Klasse `Set` in dem Modul `menge`. Implementieren Sie mindestens die folgenden Methoden.

- `add(elem)`: Hinzufügen eines Elements, wenn es noch nicht vorhanden ist.
- `union_update(seq)`: Fügt alle Elemente einer Sequenz `seq` zur Menge hinzu.
- `union(seq)`: Gibt eine neue Menge zurück, die aus den Elementen in der Menge und in der Sequenz `seq` besteht.
- `remove(elem)`: Löschen eines Elements, wenn es vorhanden ist.
- `difference_update(seq)`: Löschen aller Elemente in der Menge, die auch in der Sequenz `seq` sind.
- `difference(seq)`: Gibt eine neue Menge zurück, die aus den Elementen besteht, die in der Menge aber nicht in `seq` sind.
- `clear()`: Setze Menge auf leere Menge.
- `size()`: Gibt Anzahl der Elemente in der Menge zurück.

Es sollen die folgenden typischen Operationen in Python zur Verfügung stehen.

- Benutzen von `in` sowohl als Test als auch in einer `for`-Schleife.
- Test auf Gleichheit (`==`) und Ungleichheit (`!=`) von zwei Mengen.
- Ausgabe einer lesbaren Repräsentation auf der Kommandozeile und mit `print`.
- Verwenden der Operatoren `+` und `-` mit Mengen und Sequenzen.

Nutzen Sie für die Implementierung beliebige andere Datenstrukturen der Python-Sprache außer `set/dict` und keine Bibliotheken. Die Klasse `Set` soll sich für die implementierten Methoden so verhalten wie der eingebaute Typ `set`.

Sie können das Modul `unittest` der Standardbibliothek verwenden, um Ihre Menge zu testen. Eine mögliche Sammlung von Tests, die die Implementierung Ihrer Menge testet steht unter `test_set.py` zur Verfügung. Erweitern Sie dieses Modul mit eigenen Tests (Hinweis 2).

Aufgabe 4. Erweitern Sie das Modul `menge` der letzten Aufgabe. Realisieren Sie noch eine Klasse `OrderedSet`, die Mengen repräsentiert deren Elemente sortierbar sind. Die Klasse `OrderedSet` erbt von `Set`. Iteratoren dieser Menge iterieren dann aufsteigend sortiert über die Mengenelemente. Auch können diese Mengen miteinander verglichen werden. Es gilt die lexikographische Ordnung (Hinweis 1).

Aufgabe 5. Schreiben Sie einen Generator `gen.abwechselnd`, der zwei Generatoren nimmt und abwechselnd je aus einem Generator einen Wert nimmt und generiert.

Schreiben Sie einen Generator `pp.prim`, der alle Primzahlen generiert. Verwenden Sie diesen Generator und schreiben Sie einen weiteren Generator `pp.pprim`, der alle Primzahlpaare (zwei Primzahlen, die sich um 2 unterscheiden) generiert. Geben Sie die ersten 100 Primzahlpaare aus. Sie sollten die Funktionen in dem Modul `itertools` verwenden.

Testen Sie Ihre Generatoren mit `test_gen.py`. Erweitern Sie dieses Modul mit eigenen Tests (Hinweis 2).

Hinweis 1. Für zwei sortierte Sequenzen x, y gilt $x < y$ genau dann wenn $|x| > 0$ und $|y| = 0$, oder $x_1 < y_1$, oder $x_1 = y_1$ und x[1:] < y[1:]. Dabei steht x[1:] für die Sequenz x ohne das erste Element. Stellen Sie sich alle Elemente der Sequenz nebeneinander hingeschrieben aufsteigend sortiert vor und sortieren Sie diese Einträge dann wie in einem Telefonbuch (Zeichenketten sortieren).

Hinweis 2. Sie können sich selbst Tests schreiben unter Verwendung des Moduls `unittest` der Standardbibliothek. Die Vorlage aus `test_set.py` sollte ausreichen für die einzelnen Tests, für weitere Informationen gibt es die Online-Dokumentation. Gerade für Generatoren ist das Schreiben von Tests anspruchsvoller, da Sie darauf achten müssen nicht zwischendurch Listen aus unendlichen Generatoren zu machen. Verwenden Sie für Generatoren `itertools.islice`.

moodle Python_Barth