

Programmierung für Naturwissenschaften 1
Wintersemester 2020/2021
Übungen zur Vorlesung: Ausgabe am 06.01.2021

Aufgabe 7.1 (4 Punkte) Die *Root Mean Square*-Distanz (RMSD) wird häufig als Abstandsmaß für drei-dimensionale Strukturen, z.B. von Molekülen, verwendet. Für zwei Vektoren $\vec{v} = (v_0, v_1, \dots, v_{n-1})$ und $\vec{w} = (w_0, w_1, \dots, w_{n-1})$ mit jeweils n Punkten ist die RMSD wie folgt definiert:

$$\text{rmsd}(\vec{v}, \vec{w}) = \sqrt{\frac{1}{n} \sum_{i=0}^{n-1} (v_i.x - w_i.x)^2 + (v_i.y - w_i.y)^2 + (v_i.z - w_i.z)^2}$$

Dabei steht *.x* für die *x*-Koordinate eines Punktes. Entsprechendes gilt für *.y* und *.z*.

In Python kann man einen Punkt durch ein Triplet mit numerischen Werten (in unserem Fall Fließkommawerten) repräsentieren, sodass der erste Wert die *x*-Koordinate, der zweite Wert die *y*-Koordinate und der dritte Wert die *z*-Koordinate ist. Ein Vektor ist dann eine Liste von Punkten, d.h. eine Liste von Triplets. In den Materialien finden Sie verschiedene Dateien mit einem Vektor in jeder Zeile. Jeder Vektor ist als gültiger Python-Ausdruck dargestellt. In der Vorlesung wurde gezeigt, dass man einen String, der einen gültigen Python-Ausdruck darstellt, durch die Funktion `eval` in den entsprechenden Ausdruck konvertieren kann. Z.B. liefert

`eval('((4.0, 5.0, 6.0), (8.0, 8.0, 8.0))')` die Liste `[(4.0, 5.0, 6.0), (8.0, 8.0, 8.0)]`.

Implementieren Sie ein Python-Skript `rmsd.py`, das eine auf der Kommandozeile angegebene Datei mit Vektoren in der obigen Notation einliest und jeweils die RMSD berechnet. Genauer gesagt, soll die RMSD der beiden Vektoren in Zeile 1 und 2, der beiden Vektoren in Zeile 3 und 4, etc. bestimmt werden (siehe auch `4_vectors_rmsd.txt`). Zur Berechnung der Quadratwurzel verwenden Sie die Methode `math.sqrt`. Dazu müssen Sie das Modul `math` importieren.

Beispiel: Hier sind die beiden ersten Zeilen aus der Datei `2_vectors.txt`:

```
[(4.0, 5.0, 6.0), (8.0, 8.0, 8.0)]  
[(1.0, 2.0, 3.0), (6.0, 6.0, 6.0)]
```

Damit ist $n = 2$, $\vec{v} = (v_0, v_1)$ mit $v_0 = (4.0, 5.0, 6.0)$ und $v_1 = (8.0, 8.0, 8.0)$ sowie $\vec{w} = (w_0, w_1)$ mit $w_0 = (1.0, 2.0, 3.0)$ und $w_1 = (6.0, 6.0, 6.0)$. Damit ergibt sich die folgende Berechnung der RMSD:

$$\text{rmsd}(\vec{v}, \vec{w}) = \sqrt{\frac{1}{2}((4.0 - 1.0)^2 + (5.0 - 2.0)^2 + (6.0 - 3.0)^2 + 3 \cdot (8.0 - 6.0)^2)} = 4.41588$$

Als Lösung dieser Aufgabe müssen Sie die folgenden Funktionen implementieren:

- Eine Funktion `listofvectors_read(filename)`, die als Parameter den Namen der Datei mit den Vektoren erhält. Diese Funktion öffnet die Datei, liest sie zeilenweise ein, konvertiert mit `eval` jede einzelne Zeile in einen Vektor, d.h. eine Liste von Triplets (wie oben beschrieben), und speichert diese Vektoren nacheinander in einer Liste. Diese Liste wird durch eine `return`-Anweisung an den aufrufenden Programmteil geliefert.

- Eine Funktion `rmsd_evaluate(v_vector, w_vector)`, die zwei Vektoren als Parameter erhält und die RMSD dieser Vektoren mit einer `return`-Anweisung liefert. Verifizieren Sie in dieser Funktion, dass `v_vector` und `w_vector` die gleiche Länge haben, und dass die einzelnen Punkte, aus denen die Vektoren bestehen, Triplets sind (also genau drei Werte enthalten). Falls eine dieser beiden Bedingungen nicht erfüllt ist, muss Ihre Funktion eine sinnvolle Fehlermeldung mit `sys.stderr.write` erzeugen und mit `exit(1)` abbrechen. Die Länge einer Liste und die Anzahl der Werte wird durch die Funktion `len` bestimmt, z.B. liefert `len((1.0, 2.0, 3.0))` den Wert 3.
- Eine Funktion `listofvectors_rmsd_print(listofvectors)`, die für jedes Paar von Vektoren aus `listofvectors`, wie oben definiert, die RMSD berechnet und ihn nach der Nummer der beiden Vektoren (Zählung ab 1) in einer Zeile Tabulator-separiert mit 5 Nachkommastellen ausgibt. Beispiel: Für die beiden Vektoren aus der Datei `2_vectors.txt` muss Ihre Funktion die Zeile wie folgt ausgeben:

```
1      2      4.41588
```

Im Material finden Sie verschiedene Testdateien mit Vektoren und der erwarteten Ausgabe. Durch `make test` verifizieren Sie die Korrektheit Ihres Programms für die Testdaten.

Punkteverteilung:

- je einen Punkt für die drei Funktionen
- einen Punkt für bestandene Tests (inklusive der Tests für die Fehlerfälle).

Aufgabe 7.2 (3 Punkte) In der Datei `redundant.py` finden Sie ein Python-Programm mit vielen Redundanzen und einigen Programmteilen, die vereinfacht werden können.

Implementieren Sie in einer Datei `structured.py` eine strukturierte Version des Programms aus `redundant.py`. Die Strukturierung erfolgt durch die Deklaration einer Funktion `compare` mit folgenden Eigenschaften:

- Durch diese Funktion sollen möglichst viele Redundanzen im Programmcode der Datei `redundant.py` vermieden werden.
- Die Berechnung der Werte in den Variablen mit dem Suffix `_m` soll mit Hilfe von existierenden Python-Funktionen (siehe `python-slides.pdf`, Abschnitt Lists, frame 13) erfolgen. Dazu müssen Sie sich überlegen, welche Werte in diesen Variablen gespeichert werden.

Ihr Programm `structured.py` soll durch drei Aufrufe der Funktion `compare` mit den passenden Argumenten das gleiche Ergebnis liefern wie das Programm `redundant.py`. Das verifizieren Sie durch `make test`. Entwickeln Sie den Programmcode von `compare` schrittweise und geben Sie ggf. Zwischenergebnisse aus, die Sie mit den entsprechenden Zwischenergebnissen aus `redundant.py` vergleichen. Sie dürfen in Ihrem Programm nur das Modul `math` importieren.

Die Musterlösung umfasst 24 Zeilen Programmcode, einschließlich der Definition der drei Listen nach der Deklaration von `compare`. Ihre Lösung sollte nicht mehr als 30 Zeilen umfassen.

Hinweis zu `assert`: Im Programm stehen `assert`-Anweisungen, bei denen nach dem Schlüsselwort `assert` ein boolescher Ausdruck angegeben wird. Beim Ablauf des Programms wird jeweils verifiziert, ob dieser Ausdruck wahr ist. Wenn nicht, dann wird das Programm mit einer Fehlermeldung abgebrochen. Z.B. wird durch die Anweisung

```
assert len(x_vector) == len(y_vector) and len(x_vector) > 0
```

verifiziert, dass `x_vector` und `y_vector` gleich lang sind und mindestens ein Element enthalten.

Aufgabe 73 (3 Punkte) Ein Labormediziner möchte zukünftig Laborbefunde per Email an die beauftragenden Ärzte verschicken. Sie arbeiten bei dem Labormediziner als studentische Hilfskraft und sollen das Vorhaben technisch umsetzen.

Als Ergebnis einer Laboruntersuchung werden Dateien generiert, die die Daten in einem Tabulator-separierten Format enthalten. Hier ist ein Beispiel mit dem Inhalt einer Datei `patient1.tsv`:

```
_EMAILADRESSE_ mabuse@yahoo.com
_ANSPRACHE_ geehrter Herr Dr.
_NAME_ Mabuse
_PATIENT_ Ihres Patienten Hans Mueller
_MESSUNG_ Anzahl der Leukozyten (weiße Blutkörperchen)
_WERT_ 2000/mikro l
_BEFUND_ zu niedrig
```

Jede Zeile enthält genau einen Bezeichner einer Variablen und einen String, mit dem Wert dieser Variablen. Sie können davon ausgehen, dass der Bezeichner keine White Spaces enthält und mit einem Unterstrich beginnt und endet. Der Begriff *Variable* meint hier nicht Variablen in einem Python-Programm.

In einer weiteren Datei `email_template.txt` steht der Text einer generischen Email. Generisch bedeutet, dass der Text der Email aus festen und variablen Anteilen besteht, wie z.B. hier:

```
To: _EMAILADRESSE_
From: labormedizin@hamburg.de

Sehr _ANSPRACHE_ _NAME_,

Sie hatten uns Proben _PATIENT_ geschickt.
Die Untersuchungsergebnisse liegen nun vor.
Die _MESSUNG_ ist mit _WERT_ _BEFUND_.

Mit freundlichen Grüßen,

Ihr Labormediziner
```

Die Variablen sind die Bezeichner, die mit dem Unterstrich beginnen und enden, und deren Werte in einer anderen Datei definiert sind.

Entwickeln Sie ein Python-Programm `gen_email.py`, das beim Aufruf genau zwei Dateinamen erhält:

- den Namen einer Datei mit dem Text der generischen Email und
- den Namen einer Datei mit den Daten einer Laboruntersuchung.

Das Programm soll an allen Stellen, an denen in der generischen Email eine Variable steht, den entsprechenden String, entsprechend der Zuordnung aus der zweiten Datei ausgeben. Alle andere Worte aus der generischen Email sollen unverändert zeilenweise ausgegeben werden. Beispiel: der Aufruf von

```
./gen_email.py email_template.txt patient1.tsv
```

soll den folgenden Text im Terminal ausgeben:

To: mabuse@yahoo.com
From: labormedizin@hamburg.de

Sehr geehrter Herr Dr. Mabuse,

Sie hatten uns Proben Ihres Patienten Hans Mueller geschickt.
Die Untersuchungsergebnisse liegen nun vor.
Die Anzahl der Leukozyten (weiße Blutkörperchen) ist mit 2000/mikro l zu niedrig.

Mit freundlichen Grüßen,

Ihr Labormediziner

Das Programm soll beide Dateien nur einmal lesen. Sie müssen also Informationen aus der zweiten Datei mit den Variablen-Bezeichnern und den entsprechenden Werten in einer geeigneten Form abspeichern, um beim Lesen der ersten Datei die Ersetzungen vornehmen zu können. Um die Worte und Worttrennzeichen aufzuzählen, verwenden Sie den regulären Ausdruck `r'\w+|\W+'` und die Methode `re.findall`.

In den Materialien finden Sie Dateien mit Daten und den erwarteten Ausgaben. Durch `make test` verifizieren Sie die Korrektheit Ihres Programms.

Bitte die Lösungen zu diesen Aufgaben bis zum 11.01.2021 um 18:00 Uhr an pfn1@zbh.uni-hamburg.de schicken.