

Programmierung für Naturwissenschaften 1
Wintersemester 2020/2021
Übungen zur Vorlesung: Ausgabe am 03.02.2021

Unter dieser URL

<https://evasys-online.uni-hamburg.de/evasys/online.php?pswd=X3DGC>

oder durch Nutzung dieses QR-codes erreichen Sie die Fragebögen zur Lehrevaluation des Vorlesung und Übungen zu Programmierung für Naturwissenschaften 1.



Aufgabe 11.1 (5 Punkte) In dieser Aufgabe geht es um die Entwicklung von Klassen zur Speicherung und Verarbeitung von Moleküldaten.

Bei der computerbasierten Verarbeitung von Molekülen wird oft das mol2-Format verwendet. Eine Datei im mol2-Format kann mehrere Molekül-Einträge enthalten. Jeder Eintrag beginnt dabei mit der Header-Zeile @<TRIPOS>MOLECULE, gefolgt von einer Zeile mit dem Namen des Moleküls. Danach folgen optional Zeilen mit Informationen zum Molekül, die aber für diese Aufgabe nicht wichtig sind. In den mol2-Dateien zu dieser Aufgabe fehlen diese Zeilen.

Der mit @<TRIPOS>Atom beginnende Abschnitt listet die Atome auf, die zum Molekül gehören. Diese sind wichtig für diese Aufgabe. Ein Atom-Eintrag besteht aus 6 oder mehr Werten, die in einer Zeile vorliegen: einer ID, einem Namen, den Koordinaten (x, y, z) und dem Atomtyp. Es folgen weitere optionale Werte.

Ein weiterer Abschnitt nach @<TRIPOS>Bond listet die Bindungen zwischen Atomen im Molekül auf. Ein Eintrag enthält die ID der Bindung, zwei IDs Atom-ID₁ und Atom-ID₂ der gebundenen Atome und die Art der Bindung (1 = einfach, 2 = zweifach, ar = aromatisch). Es folgen weitere optionale Werte.

Hier ein Beispiel eines mol2-Eintrags für ein bekanntes Molekül mit dem Namen water:

```
@<TRIPOS>MOLECULE
water
@<TRIPOS>ATOM
  1 O          -0.5253   -0.0510   -0.3145  0.3      1 HOH1      -0.4105
  2 H          -0.9420    0.7479    0.0113  H        0 HOH0       0.2052
  3 H           0.4037    0.0598   -0.0736  H        0 HOH0       0.2052
@<TRIPOS>BOND
  1      1      3      1
  2      1      2      1
```

In den Materialien finden Sie eine Datei `molecule_template.py`. Bitte benennen Sie diese in `molecule.py` um. Sie sollen das in der Datei vorhandene Gerüst der Klassen `Molecule`, `Atom` und `Bond` so vervollständigen, dass die Unittests bestanden werden und das Programm `mol2iter.py` mol2-Dateien verarbeitet und die oben beschriebene Information im gleichen Format (abgesehen von der Anzahl der Leerzeichen) wieder ausgibt. Die Methoden, die mit einem Kommentar der Form `required only for` versehen sind, brauchen Sie nicht zu implementieren.

Im Material finden Sie mol2-Dateien, sowie die beiden genannten Python-Dateien. `mol2iter.py` enthält bereits eine vollständige Funktion `mol2Iterator` zum Einlesen einer mol2-Datei, die Sie nicht verändern dürfen. In den Zeilen

```
for name, atom_list, bond_list in mol2Iterator(mol2file):  
    molecule_list.append(Molecule(name, atom_list, bond_list))
```

wird ein mol2-Eintrag aus der angegebenen Datei gelesen und der Name des Moleküls sowie die Liste der Atome und die Liste der Bindungen zurückgeliefert. Jedes Element aus `atom_list` ist selbst wieder eine Liste mit 6 oder mehr Werten, nämlich den Werten einer Atomzeile. Jedes Element aus `bond_list` ist selbst wieder eine Liste mit 4 oder mehr Werten, nämlich den Werten einer Bondzeile.

Die Ausgabe der Moleküle erfolgt in den folgenden Zeilen:

```
for molecule in molecule_list:  
    print('{}'.format(molecule))
```

Die Klasse `Molecule` soll den Namen eines Moleküls sowie die Liste von Atomen und die Liste von Bindungen speichern. Die Atome und Bindungen sind jeweils Instanzen der Klasse `Atom` bzw. `Bond`. Die `__init__`-Methode der Klasse `Molecule` erhält dazu (nach dem Parameter `self`) den Molekülnamen sowie die Listen aller Atome und Bindungen des Moleküls (siehe oben) und sie muss diese in entsprechenden Instanzvariablen¹ `self._name`, `self._atom_list`, `self._bond_list` speichern. Entsprechende Methoden `name(self)`, `atom_list(self)` und `bond_list(self)`, die die Werte der drei Instanzvariablen zurückliefern sind bereits implementiert. Sie müssen noch die `__str__`-Methode der Klasse `Molecule` implementieren. Diese Methode soll einen String im mol2-Format (inklusive der verschiedenen Headerzeilen der Form `@<TRIPOS> . . .`) zurückliefern.

Neben der Klasse `Molecule` müssen noch die Klassen `Atom` und `Bond` implementiert werden, die die Werte für ein Atom bzw. eine Bindung speichern.

Für ein Atom muss es Instanzvariablen für eine Atom-ID, einen Namen, drei Koordinaten als Fließkommawerte, einen Atomtyp sowie eine Liste weiterer optionaler Werte geben. Entsprechend hat die Methode `__init__` der Klasse `Atom` nach `self` noch 7 weitere Parameter und entsprechende Instanzvariablen. Die `__str__`-Methode der Klasse `Atom` gibt einen String zurück, der aus den Werten besteht, die ein Atom beschreiben.

Für eine Bindung aus der Klasse `Bond` muss es Instanzvariablen für die ID der Bindung, die Atom-IDs der an der Bindung beteiligten Atome sowie den Bindungstyp und optionale Angaben geben. Die optionalen Werte einer Zeile werden als Liste übergeben. Entsprechend hat die `__init__`-Methode der Klasse nach `self` noch 5 Parameter, und es gibt mindestens 5 Instanz-Variablen in der Klasse. Die `__str__`-Methode der Klasse `Bond` gibt einen String zurück, der aus den Werten besteht, die die Bindungen beschreiben.

¹Dieser Begriff ist ein Synonym für Member-Variable.

Beachten Sie, dass die `__str__`-Methoden der drei zu implementierenden Klassen jeweils eine String-Repräsentation einer Instanz der Klasse zurückliefert, die (abgesehen von Leerzeichen) der Darstellung des mol2-Format entspricht. Dadurch kann man z.B. ein Molekül durch eine Anweisung `print('{}'.format(molecule))` formatiert ausgeben. Innerhalb einer Zeile einer der genannten String-Repräsentationen werden aufeinanderfolgende Werte jeweils durch genau ein Leerzeichen getrennt. Die Reihenfolge der Zeilen und der Werte innerhalb einer Zeile entspricht der Reihenfolge in der Eingabedatei. Dadurch lassen sich die `__str__`-Funktionen auf einfache Weise testen. Für die Klassen `Atom` und `Bond` gibt es entsprechende Unittests.

Diese Beschreibung ist so ausführlich geworden, damit Sie selbst nicht das Design der Klassen entwickeln müssen. Ihre Aufgabe ist es, diese Spezifikation zu implementieren und zwar in der Datei `molecule.py`.

In späteren Übungsaufgaben werden Sie die hier genannten Klassen um weitere Methoden und weitere Klassen ergänzen.

Aufgabe 11.2 (5 Punkte) In allen wissenschaftlichen Bereichen entstehen vielfältige Daten, die Beziehungen zwischen Datenpunkten durch numerische Werte beschreiben. So wird die in der medizinischen Chemie die Effizienz von Wirkstoffen in Bezug auf Pathogene quantifiziert. In der Genomforschung werden Expressionsänderungen von Genen einer Spezies unter Einwirkung verschiedener Stressfaktoren gemessen. In der Physik wird die Wechselwirkung von Atomen unter extremen physikalischen Bedingungen untersucht. In der Bioinformatik wird die Ähnlichkeit von Sequenzen berechnet.

In dieser Aufgabe soll eine Klasse zur Verwaltung von Ergebnissen paarweiser Vergleiche entwickelt werden. Die Ergebnisse sind numerische Werte (im folgenden Scores genannt). Die untersuchten Objekte werden jeweils alle durch einen eindeutigen Bezeichner identifiziert. Das Ziel ist, die Daten mit den Methoden der Klasse aufzubereiten, so dass man daraus mit wenig Aufwand eine sogenannte Heatmap erzeugen kann. Diese stellt Scores durch Farben auf einer dynamisch angepassten Farbskala (`colormap`) dar.

Benennen Sie die Datei `draw_pairwise_heatmap_template.py` aus den Materialien in `draw_pairwise_heatmap.py` um. Implementieren Sie in der Klasse `PairwiseScores` die folgenden Methoden. Natürlich hat diese Klasse auch Member-Variablen, auf die jeweils nicht ausserhalb der Klasse zugegriffen werden darf. Daher verwenden Sie für die einzelnen Namen der Member-Variablen Bezeichner, die mit einem Unterstrich beginnen.

1. Die Methode `__init__(self, inputfile)` öffnet eine Datei, deren Namen durch den Parameter `inputfile` übergeben wird. Die geöffnete Datei darf nur einmal gelesen werden und Sie enthält zwei Formen von Zeilen:
 - Zeilen, die jeweils mit dem Zeichen `#` beginnen enthalten beliebigen Text. Die erste solche Zeile nach Löschen dieses Zeichens und der direkt folgenden Leerzeichen enthält den Titel der erzeugten Heatmap. Alle anderen Zeilen dieser Form werden ignoriert.
 - Zeilen, die nicht mit dem Zeichen `#` beginnen enthalten die Daten in Form von drei jeweils durch einen Tabulator getrennten Werten. Die ersten beiden Werten sind Schlüssel, nämlich der `row_key` (erster Wert) und der `column_key` (zweiter Wert). Der dritte Wert ist ein numerischer Wert (der Score), den Sie durch einen `float` Wert repräsentieren. Die Schlüssel und Schlüsselpaare müssen nicht eindeutig sein. Wenn ein Schlüsselpaar mehrmals Mal auftaucht, wird der bisherige Wert überschrieben.

2. Die Methode `row_keys(self)` liefert die lexikografisch sortierte Liste der `row_keys` zurück.
3. Die Methode `column_keys(self)` liefert die lexikografisch sortierte Liste der `column_keys` zurück.
4. Die Methode `lookup(self, rk, ck)` liefert den Score für den `row_key rk` und den `column_key ck`.
5. Die Methode `title(self)` liefert den Titel zurück. Falls es in der Eingabedatei keine Titelzeile gab, wird der Wert `None`.
6. Die Methode `np_matrix(self)` liefert eine `numpy-Matrix self._pws` mit m Zeilen und n Spalten, wobei m die Anzahl der `row_keys` und n die Anzahl der `column_keys` ist. Beachten Sie, dass Sie die Matrix mit `self._pws = np.empty((m,n))` erst dann erzeugen, nachdem alle Werte eingelesen wurden. Denn erst dann kennen Sie die Werte von m und n . Sie müssen also die eingelesenen Werte in geeigneter Weise in einer Member-Variable speichern, in die sie mit jeder neuen Datenzeile Werte eintragen.

Sei a der i -te Schlüssel in `row_keys` und b der j -te Schlüssel in `column_keys`. Falls es eine Zeile mit den Schlüsseln a und b und dem Score s gibt, dann steht in `self._pws[i, j]` der Wert s , sonst `np.NaN`. Es ist sinnvoll, nach dem Erzeugen der `numpy-Matrix` diese mit `self._pws.fill(np.NaN)` zu initialisieren. Ermitteln Sie, wie sich die Verwendung von `np.NaN` für nicht-vorhandene Scores auf die erzeugte Heatmap auswirkt und beschreiben Sie kurz den Effekt als Kommentar zur Implementierung dieser Methode.

$\frac{1}{2}$ Punkt

In `draw_pairwise_heatmap.py` finden Sie einen Optionsparser und ein Hauptprogramm, sowie Unittests. Durch `make test` verifizieren Sie, dass die Unittests bestanden werden. Ausserdem wird, wenn Ihre Methoden korrekt implementiert wurden, eine PDF-Datei mit einer Heatmap erzeugt, siehe Beispiel in Abbildung 1.

Punkteverteilung:

- Implementierung der Klasse: 3.5 Punkte
- Beantwortung der Frage zu `np.NaN`: 0.5 Punkte
- Bestandene Tests: 1 Punkt

Bitte die Lösungen zu diesen Aufgaben bis zum 08.02.2021 um 18:00 Uhr an pfn1@zbh.uni-hamburg.de schicken.

Abbildung 1: Heatmap zur Darstellung der Daten aus der Datei `alignment_scores.tsv` im Verzeichnis mit den Materialien für Aufgabe 111.2.

