

**Programmierung für Naturwissenschaften 1
Wintersemester 2020/2021
Übungen zur Vorlesung: Ausgabe am 13.01.2021**

Aufgabe 8.1 (2 Punkte)

Aus Zeitgründen können wir in diesem Semester in der Vorlesung das Thema Codon-Translation (siehe Folien Abschnitt 18, frame 1-8) nicht behandeln. Aus der Schule kennen Sie bereits die biologischen Grundlagen und in unserem Modul haben Sie die notwendigen Python-Kenntnisse erworben, um das Thema zu verstehen. Für diese Übungsaufgabe müssen sich einige Studierende vor der Übung anhand der Vorlesungsfolien auf dieses Thema vorbereiten und das erworbene Wissen in Kleingruppen in den ersten 30 Minuten der Übung an die anderen Studierenden weitergeben.

Die Kleingruppen setzen sich wie folgt zusammen:

- | | |
|---|---|
| 1. Nichvolodina, Tchonnet Toukam, Vehns | 9. Fuchs, Jakovljevic, Lim |
| 2. Hossfeld, Kourouklis, Schimansky | 10. Koerper, Pohle, Rodrigues Nobre |
| 3. Mehammed, Nergiz, Stephan | 11. Maecking, Riemann, Schulta |
| 4. Koester, Coin, Wang, Schindler, Unland | 12. Nachtschatt, Naehring, Neuhaus |
| 5. Kemnitz, Le, Trigkas Chatziandreou | 13. Merl, Sakhire, von der Osten-Sacken |
| 6. Jolic, Zozanyan, Beiersdorf | 14. ASchroeder, Krupp, Kroeger, Spurny |
| 7. Hiep, Reich, Wacke | 15. Altmann, Chow Castro, Fikani |
| 8. Albetani, Liebsch, Mrozek | 16. Buchstab, Duez, Koegel |

Die Namen der Studierenden, die sich auf das Thema vorbereiten müssen, sind jeweils am Beginn jeder Namensliste aufgeführt. Falls jemand von diesen Studierenden nicht zur Übung erscheint, werden die übrigen Mitglieder der betroffenen Kleingruppe am Anfang der Übung auf die anderen Kleingruppen verteilt.

Diese Aufgabe soll am Anfang der Übung bearbeitet werden. Dazu erarbeiten die Kleingruppen zusammen die Folien zum Thema Codon-Translation. Der oder die Studierende, die sich vorbereitet hat, kann ggf. bei Unklarheiten Fragen, zumindest zu den Grundbegriffen, beantworten oder Erläuterungen mündlich ergänzen. Falls Sie die in den Folien dargestellte auf einen dictionary basierte Implementierung ausprobieren möchten, können Sie auf den Programmcode aus den Materialien zurückgreifen.

Nach der Übung dokumentiert jede Kleingruppe in einer E-mail an `pfn1@zbh.uni-hamburg.de` das Vorgehen bei der Erarbeitung des Themas und ggf. noch bestehende Verständnisfragen oder Hinweise zu Unklarheiten in den Folien. Willkommen sind natürlich auch Bemerkungen zur Lehrform selbst. Dabei soll nicht der Inhalt der Folien repliziert werden. Die E-mail soll die folgenden Eigenschaften haben:

- die Betreffzeile lautet `codontransPeer` in genau dieser Schreibweise,
- abgesendet bis zum Abgabetermin der entsprechenden Übung,

- maximal 15 Zeilen mit maximal 80 Zeichen pro Zeile,
- Angabe der Nachnamen aller Mitglieder der Kleingruppe, die teilgenommen haben (alle genannten Personen erhalten die zwei Punkte),
- keine Anhänge.

Die E-mail soll von einer/einem Studierenden erstellt werden, die/der sich nicht vorher auf das Thema vorbereitet hat.

Aufgabe 8.2 (6 Punkte) In dieser Aufgabe geht es um die Generierung von Passwörtern. Einerseits sollen Passwörter sicher sein, d.h. man soll sie nur sehr schwer erraten können. Andererseits soll man sich Passwörter einfach merken können.

Eine häufig verwendete Methode mit diesen Eigenschaften generiert Passwörter entsprechend einer vorgegebenen Struktur aus Ziffern, Satzzeichen und Worten, jeweils mit einer vorgegebenen Anzahl bzw. Länge. In dieser Aufgabe besteht ein *Strukturelement* aus einem der Zeichen `d`, `p` und `w`, jeweils gefolgt von einer ganzen Zahl. So steht `d3` für drei zufällig ausgewählte Ziffern, `p4` steht für 4 zufällig ausgewählte Satzzeichen und `w5` steht für ein zufällig ausgewähltes Wort der Länge 5 aus einer vorgegebenen Liste von Worten.

Beispiel: Der Strukturstring `w4p2d1w5` beschreibt alle Passwörter, die sich in dieser Reihenfolge wie folgt zusammensetzen:

- ein Wort der Länge 4,
- zwei Satzzeichen,
- eine Ziffer,
- ein Wort der Länge 5.

Hier sind einige Passwörter mit dieser Struktur dargestellt. Dabei wird die Wortliste mit insgesamt 23 374 Worten aus dem Material zu dieser Aufgabe zu Grunde gelegt.

```
meet!`7rosso
dory]\6clish
sump/^6ethyl
fate+?9wrung
tomb.:9ginny
```

In dieser Aufgabe sollen zunächst vier Funktionen implementiert werden, die in einer Aufgabe des nächsten Übungsblatts in einem lauffähigen Programm verwendet werden sollen. Implementieren Sie in der Datei `pwgen_functions.py` die folgenden Funktionen:

- Die Funktion `structure_string_is_valid(struct_str)` liefert mit einer `return`-Anweisung genau dann `True` zurück, wenn der Strukturstring `struct_str` syntaktisch korrekt geformt ist, d.h. aus einem oder mehreren Strukturelementen (siehe oben) besteht. Verwenden Sie in Ihrer Implementierung einen regulären Ausdruck und die passende Methode, um zu verifizieren, dass der Strukturstring zum reguläre Ausdruck passt. Bedenken Sie dabei, dass der Ausdruck mit `^` und `$` verankert werden muss.
- Der Generator `structure_elements_enumerate(struct_str)` zerlegt die einzelnen Strukturelemente und generiert Paare, bestehend aus einem String mit jeweils genau einem

der drei genannten Zeichen und der entsprechenden ganzen Zahl. Generieren von Paaren heißt in diesem Kontext, mit `yield` jedes Paar an den aufrufenden Programmteil zu liefern.

Beispiel: Für den Strukturstring `w4p2d1w5` generiert der Generator nacheinander `('w', 4)`, `('p', 2)`, `('d', 1)`, `('w', 5)`.

Verwenden Sie zur Implementierung dieser Funktion einen regulären Ausdruck (RE) für ein einzelnes Strukturelement und die passende Funktion aus dem Modul `re`, um die Treffer des RE im Strukturstring aufzuzählen. Aus den Treffern generieren Sie dann die Paare, wie oben beispielhaft gezeigt. Beachten Sie, dass das zweite Element eines zu generierenden Paares eine ganze Zahl ist, so dass der extrahierte String mit `int` konvertiert werden muss.

- Die Funktion `randstring(alphabet, n)` liefert in einer `return`-Anweisung einen zufälligen String der Länge `n` über dem Alphabet `alphabet` (repräsentiert als String). Die einzelnen Zeichen des zufälligen Strings sollen mit der gleichen Wahrscheinlichkeit unabhängig voneinander generiert werden. Verwenden Sie dazu die Methode `randint` aus dem Modul `random`, siehe auch `python-slides.pdf`, Abschnitt *Abstract data types and classes*, frame 14-15. Ein zufälliges Zeichen aus `alphabet` berechnet man durch den Ausdruck

```
alphabet[random.randint(0, alpha_size-1)]
```

wobei `alpha_size` die Anzahl der Zeichen in `alphabet` ist. Um einen zufälligen String der Länge `n` zu erhalten, muss man diesen Ausdruck `n` mal auswerten und die gelieferten Zeichen in einem String zusammenfassen.

- Die Funktion `word_dict_get()` öffnet die Datei `wordlist.txt` und liest die darin enthaltenen Worte zeilenweise ein. Mit einer `return`-Anweisung wird ein Dictionary `wd` zurückgeliefert, so dass `wd[m]` die Liste der Worte der Länge `m` aus dieser Datei ist. Falls es in `wordlist.txt` kein Wort einer Länge `m` gibt, dann ist `m` kein Schlüssel in `wd`.

Für die Funktion sind in der Datei `pwgen_functions_unit_test.py` sogenannte Unit-Tests implementiert. Durch `make test` werden diese Unit-Tests für Ihre Implementierung der vier Funktionen ausgeführt.

Teilaufgabe: Beschreiben Sie in wenigen Sätzen als Kommentar in Ihrer Implementierungsdatei, woraus diese Unit-Tests bestehen. Hierzu müssen Sie ggf. recherchieren, was z.B. `assertEqual` bedeutet. Nachdem Sie die Funktionen implementiert haben, verifizieren Sie durch `make test` die Korrektheit Ihrer Implementierung.

Punkteverteilung:

- jeweils ein Punkt für die vier Funktionen
- 1 Punkt für die Beschreibung der Unit-Tests
- 1 Punkt für alle bestandenen Unit-Tests

Aufgabe 8.3 (2 Punkte) In der Vorlesung wurde im Abschnitt *Reading and representing data matrices* gezeigt, wie man eine Matrix in ein Dictionary von Dictionaries konvertiert und dieses wieder ausgibt. Den entsprechenden Programmcode finden Sie in den beiden Dateien `data_matrix.py` und `data_matrix_main.py` im Verzeichnis `pfn1_2020/Vorlesung/src/Chemistry`.

Implementieren Sie nun in einer Datei `data_matrix_class.py` eine Klasse `DataMatrix`, die die gleiche Funktionalität bietet, wie die drei Funktionen aus `data_matrix.py`. Im Einzelnen müssen Sie die genannte Klasse mit zwei Member-Variablen `_matrix` und `_attribute_list` und den folgenden Methoden implementieren:

- `__init__(self, lines, key_col=1, sep='\t')` entspricht der Methode `data_matrix_new` aus `data_matrix.py`. `__init__` liefert natürlich keine Werte über eine `return`-Anweisung, sondern speichert die Matrix und die Attributliste in den genannten Member-Variablen.
- Die Methode `show(self, sep, attributes, keys)` soll das Gleiche leisten, wie die Funktion `data_matrix_show`.
- Die Methode `show_orig(self, sep, attributes, keys)` soll das Gleiche leisten, wie die Funktion `data_matrix_show_orig`.
- Die Methode `keys(self)` liefert `self._matrix.keys()` über eine `return`-Anweisung.
- Die Methode `attribute_list(self)` liefert `self._attribute_list` über eine `return`-Anweisung.

Die beiden Member-Variablen sind privat und dürfen nicht außerhalb der Klassendefinition verwendet werden.

Nach der Implementierung der Klasse kopieren Sie den Programmcode aus `data_matrix_main.py` in die Datei `data_matrix_class.py` und modifizieren ihn so, dass die Funktionalität bzgl. der Datenmatrizen durch die Methoden der Klasse `DataMatrix` realisiert wird. Das Hauptprogramm soll ausgeführt werden, wenn `__name__ == '__main__'` gilt. Dadurch kann die Klasse in Zukunft weiterverwendet werden.

In den Materialien finden Sie zwei Testdateien `atom-data-mini.tsv` und `atom-data.tsv` und ein Makefile. Durch `make test` verifizieren Sie, dass Ihr Programm für diese Testdaten korrekt funktioniert.

Bitte die Lösungen zu diesen Aufgaben bis zum 18.01.2021 um 18:00 Uhr an pfn1@zbh.uni-hamburg.de schicken.

Effizienz der Methoden zur Translation von Codons

In den Folien zum Thema Codontranslation wird gesagt, dass die Methode, die ein Dictionary verwendet, die effizienteste ist. Es wurde gefragt, warum das so ist und was genau Effizienz in diesem Zusammenhang bedeutet.

Ich will hier kurze Antworten geben. Vielleicht mache ich es noch zu einem Thema einer Peer-Teaching Aufgabe. Das Thema Effizienz von Algorithmen wird im Modul Algorithmen und Datenstrukturen genau betrachtet.

Wenn man bei Algorithmen von Effizienz spricht, meint man meist die Laufzeit oder den Speicherbedarf. Beim Speicherbedarf zählt immer die maximale Größe, die der Algorithmus während der Laufzeit benötigt. Laufzeit und Speicher werden durch Funktionen, die abhängig von der Eingabegröße sind, ausgedrückt.

Wenn sich z.B. die Anzahl der Schritte eines Algorithmus für eine Eingabe der Größe n durch die Funktion $h(n) = 500$ ausdrücken lässt, dann spricht man von konstanter Laufzeit. Wenn sich z.B. die Anzahl der Schritte eines Algorithmus für eine Eingabe der Größe n durch die Funktion $f(n) = 20 + 5n$ berechnen lässt, sagt man: Die Laufzeit ist linear. Wenn sich die Anzahl der Schritte eines Algorithmus durch die Funktion $g(n) = 1000 + 100n + 2n^2$ berechnen lässt, dann spricht man von quadratischer Laufzeit. Es zählt also immer nur der dominierende Term. Damit werden

Faktoren und Konstanten ignoriert.

Im Fall der Codon Translation ergibt sich für alle drei Varianten aus den Folien eine konstante Laufzeit pro Codon und damit eine lineare Laufzeit für die gesamte Sequenz. Der Zugriff auf ein Dictionary für einen gegebenen Schlüssel benötigt nur wenige Rechenschritte, da nur der Schlüssel (also ein Codon) in einem numerischen Wert konvertiert werden muss, aus dem man mit einer Rechenoperation eine Adresse im Speicher berechnen kann. Für beide Schritte benötigt man auf einer modernen CPU nur wenige Rechenschritte.

Bei der ersten Variante muss man im Schnitt $\frac{64}{2} = 32$ Vergleiche durchführen, um festzustellen, welcher der 64 Fälle zutrifft. Dafür benötigt man sehr viel mehr Rechenschritte als bei der dritten Variante. Bei der Variante mit den regulären Ausdrücken wird dieser in einen sogenannten endlichen Automaten transformiert, mit dem die Sequenz durchsucht wird. Das benötigt weniger Rechenschritte als die erste Variante, aber sicher mehr als die dritte Variante. Die genaue Anzahl der Schritte für die einzelnen Methoden lässt sich nur sehr schwer bestimmen und sie hängt von der Programmiersprache ab, von der Art und Weise, wie in der Programmiersprache die genannten Datenstrukturen (Dictionaries, endliche Automaten) implementiert sind, vom Betriebssystem und dem Rechner. Daher müsste man konkrete Messungen durchführen, um den Unterschied in der Praxis zu messen.