Stefan Kurtz **Universität Hamburg**
Jeanine Liebold **Zentrum für Bioinformatik**

Programmierung für Naturwissenschaften 1
Wintersemester 2022/2023
Übungen zur Vorlesung: Ausgabe am 07.12.2022

Selbsttest-Aufgaben zum Thema Kontrollstrukturen, Listen, Strings und Dictionaries in Python

Um ihre eigenen Fähigkeiten in der Python-Programmierung zu testen, lösen Sie bitte selbständig die folgenden Aufgaben. Nutzen Sie zunächst möglichst nicht den Python-Interpreter, sondern nur Papier und Stift. Sie können Ihre Lösung natürlich später mit dem Python-Interpreter verifizieren. Ihre Lösungen geben Sie nicht ab. Nutzen Sie die ersten 15 Minuten der Übung für diese Aufgaben.

1. Geben Sie bitte an, was das folgende Programmstück ausgibt.

```
values = [(2,'C'),(-1,'A'),(-2,'C'),(4,'A'),(-1,'D'),
          (0,'B'),(-1,'A'),(1,'A'),(-2,'A'),(-2,'T')]
result = None
for i, cc in values:
  if result is None or i < result[0]:
    result = (i,[cc])
  elif i == result[0]:
    result[1].append(cc)
print(''.join(result[1]))
```

2. Was gibt das folgende Programm aus? Umschreiben Sie das Ergebnis für den Fall, dass `sequence` eine beliebige Sequenz mit Zeichen über dem Alphabet {`a`, `c`, `g`, `t`} ist.

```
sequence = 'actatgaggta'
alphabet = 'acgt'
d2 = dict()
for x in alphabet:
  for y in alphabet:
    d2['{}{}'.format(x,y)] = len(d2)
occ = [0] * len(d2)
for i in range(len(sequence)-1):
  occ[d2[sequence[i:i+2]]] += 1
for j, occv in enumerate(occ):
  if occv > 0:
    print('{}{} {}'.format(alphabet[j//4],alphabet[j % 4],occv))
```

3. Schreiben Sie das vorherige Programm so um, dass es (abgesehen von der Reihenfolge der Zeilen) die gleiche Ausgabe liefert, dazu aber nur genau ein Dictionary verwendet und keine Liste.

4. Betrachten Sie die folgenden Funktion. Welche Anforderung für `f` ergibt sich aus dem Kopf der `for`-Schleife? Welche weitere Anforderung für `f` ergibt sich aus dem Rumpf der `for`-Schleife? Welche Eigenschaft hat `g`, wenn die `return`-Anweisung ausgeführt wird?

```
def from_f_to_g(f):
  g = dict()
  for k, v in f.items():
    g[v] = k
  return g
```

5. Im folgenden Programm wird in der letzten Anweisung die Variable `role` verwendet. Diese wird aber nicht in dem für Sie sichtbaren Teil des Programms definiert. Welchen Wert sollte `role` haben, damit die Aussagen, die durch `print` ausgegeben werden, sinnvoll sind. Geben Sie diese Aussagen an. Sie können davon ausgehen, dass die im Programm verwendeten Bezeichner sinnvoll gewählt wurden.

```
parents_of = dict()
for kid, father, mother in [('Hans','Fritz','Erna'),\
                            ('Fritz','Willi','Edda'),\
                            ('Erna','Eberhardt','Ulla')]:
  parents_of[kid] = [father,mother]
for kid, parents in parents_of.items():
  for p in parents:
    if p in parents_of:
      for q in parents_of[p]:
        print('{} ist {} von {}'.format(kid,role,q))
```

Aufgabe 7.1 (2 Punkte)

Aus Zeitgründen können wir in diesem Semester in der Vorlesung das Thema Codon-Translation (siehe Folien Abschnitt 20, 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 muss sich aus jeder Kleingruppe ein Student bzw. eine Studentin anhand der Vorlesungsfolien auf dieses Thema vorbereiten. Sie sollten untereinander klären, wer diese Vorbereitung in dieser Woche übernimmt. In den kommenden Übungen wird es weitere Aufgaben dieses Typs geben, so dass jeder Studierende sich mindestens einmal vorbereiten muss.

Das in der Vorbereitung erworbene Wissen soll an die anderen Mitglieder der jeweiligen Kleingruppe dadurch weitergegeben werden, dass man zusammen den Inhalt der oben genannten Folien zum Thema Codon-Translation bespricht. Das kann entweder in den ersten 15-20 Minuten der Übung erfolgen oder zu einem anderen Zeitpunkt außerhalb des Übungstermins, falls z.B. ein Studierender nicht an der Übung teilnehmen kann.

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 einem Dictionary basierte Implementierung ausprobieren möchten, können Sie auf den Programmcode aus den Materialien zurückgreifen. Die Antwort auf eine Frage eines Studierenden zur Effizienz der Codon-Translation finden Sie am Ende des Übungsblattes.

Nach Bearbeitung dieser Aufgabe dokumentiert jede Kleingruppe in der Textdatei `bearbeitung.txt` in höchstens 15 Zeilen mit maximal 80 Zeichen pro Zeile ihr 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. Es soll nicht der Inhalt der Folien wiedergegeben werden. Selbstverständlich müssen Sie in der Datei auch die üblichen Metadaten angeben.

Aufgabe 7.2 (4 Punkte)

Wir betrachten ein Alphabet der Größe r mit den Zeichen $a_1, a_2, \ldots, a_r$. Für alle i, $0 \leq i \leq r$ ist der i-te Skyline-String s_i wie folgt definiert:

$$s_i = \begin{cases} \varepsilon & \text{falls } i = 0 \\ s_{i-1} a_i s_{i-1} & \text{sonst} \end{cases}$$

Hier steht ε für den leeren String der Länge 0.

Beispiel: Für das Alphabet $\{a, b, c, d\}$ ergeben sich die folgenden Skyline-Strings $s_1 = a$, $s_2 = aba$, $s_3 = abacaba$ und $s_4 = abacabadabacaba$.

Hintergrund: Für bestimmte Algorithmen zur Sortierung von Strings bilden Skyline-Strings die Eingabe, die zur maximalen Laufzeit führt. Dieser Aspekt spielt in dieser Aufgabe jedoch keine Rolle.

Bitte gehen Sie wie folgt vor:

- Geben Sie in der Datei `skyline_length.txt` an, welche Länge s_i hat. Sie sollen hier keine konkreten Zahlen angeben, sondern einen nicht-rekursiven mathematischen Ausdruck, durch den, abhängig von i, die Länge von s_i für beliebige i, bestimmt werden kann. Sie können versuchen, einen solchen Ausdruck direkt aus der Definition von s_i abzuleiten, oder aus den Längen der Skyline-Strings in der Datei `skyline1_9.txt`.
- Wir setzen das Alphabet $\{a, b, c, \ldots, z\}$ von 26 Kleinbuchstaben voraus, d.h. $r = 26$. Implementieren Sie in der Datei `skyline.py` eine Python-Funktion `skyline`, die genau einen ganzzahligen Parameter i erhält und den $i$ten Skyline-String mit einer `return`-Anweisung zurückliefert. Dabei müssen Sie das i-te Zeichen des obigen Alphabets mit Hilfe der Methoden `ord` und `chr` berechnen.
- Implementieren Sie in der genannten Python-Datei eine Funktion `parse_command_line`, die als Argument eine Liste von Strings erhält und Folgendes überprüft und ggf. Paare von zwei ganzen Zahlen zurückliefert.
  - Die übergebene Liste muss die Länge 1 oder 2 haben. Falls das nicht gilt, muss eine passende `Usage`-Zeile ausgegeben und mit `exit(1)` abgebrochen werden.
  - Falls die übergebene Liste die Länge 1 hat, muss das einzige Listenelement eine positive ganze Zahl $i \leq 26$ repräsentieren und die Funktion liefert das Paar (i, i) mit einer `return`-Anweisung zurück.
  - Falls die übergebene Liste die Länge 2 hat, müssen die beiden Listenelemente zwei positive ganze Zahlen ℓ und u (jeweils ≤ 26) repräsentieren und die Funktion liefert das Paar (ℓ, u) mit einer `return`-Anweisung zurück.

  Behandeln Sie fehlerhafte Strings[1] in der übergebenen Liste mit Hilfe der in der Vorlesung vor-

[1] Fehlerfälle: Umwandlung in ganze Zahlen ist nicht möglich oder ganze Zahl ist nicht im geforderten Wertebereich

gestellten Techniken. Im Falle eines Fehlers geben Sie jeweils eine sinnvolle Fehlermeldung aus und rufen `exit(1)` auf. Im Falle einer Ausnahmebehandlung soll die Fehlermeldung des Python-Interpreters mit ausgegeben werden.

- Vermeiden Sie soweit wie möglich redundanten Programmcode, indem Sie Funktionen einführen oder Programmcode in Schleifen einbetten.

Das Hauptprogramm ruft nun zuerst die Funktion `parse_command_line` mit dem Argument `sys.argv[1:]` auf. Die Funktion liefert das Paar (ℓ, u) zurück, wenn sie nicht aufgrund eines Fehler das Programm mit `exit(1)` abgebrochen hat. Es soll für alle i, $\ell \leq i \leq u$ jeweils der i-te Skyline-String ausgegeben werden. Vergessen Sie in Ihrem Hauptprogramm nicht den magic-String und das Setzen der Ausführungsrechte.

Im Material zu dieser Übung finden Sie die erwartete Ausgabe Ihres Programms sowie ein Makefile. Durch `make test` können Sie verifizieren, ob Ihr Programm für einige Testaufrufe korrekt funktioniert.

Punkteverteilung:

- 1 Punkt für die korrekte Aussage zur Länge der Skyline-Sequenzen, abhängig von i.
- 1 Punkte für eine nachvollziehbare Implementierung der Funktion `skyline`.
- 1 Punkte für eine nachvollziehbare Implementierung der Funktion `parse_command_line` inklusive der Fehlerbehandlung.
- 1 Punkt für das Bestehen der Tests.

Aufgabe 7.3 (3 Punkte)

Eine Primzahl ist eine ganze Zahl $p \geq 2$, die sich nicht ganzzahlig durch eine kleinere Zahl $q \geq 2$ teilen lässt. In dieser Aufgabe soll der Algorithmus „Sieb des Eratosthenes“ zur Generierung einer Liste von Primzahlen $\leq n$ implementiert werden. n ist dabei ein von der Benutzerin definierter Wert. Der Algorithmus funktioniert wie folgt:

Man notiert zunächst alle Zahlen i, $2 \leq i \leq n$. Z.B. ergibt sich für $n = 20$ die folgende Zahlenfolge

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Im nächsten Schritt werden alle echten Vielfachen von 2 markiert (hier durch das Symbol $\times$ dargestellt).

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
		$\times$		$\times$		$\times$		$\times$		$\times$		$\times$		$\times$		$\times$		$\times$

Es folgt die zusätzliche Markierung aller echten Vielfachen der kleinsten unmarkierten Zahl > 2, also der 3:

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
		$\times$		$\times$		$\times$	$\times$	$\times$		$\times$		$\times$	$\times$	$\times$		$\times$		$\times$

Es folgt die zusätzliche Markierung aller echten Vielfachen der kleinsten unmarkierten Zahl > 3, also der 5. Da alle Vielfachen von 5, die kleiner als 20 sind, bereits markiert wurden, ergibt sich keine neue Markierung.

In jedem Schritt werden also jeweils die echten Vielfachen der nächsten noch unmarkierten Zahl zusätzlich markiert. Das wird solange wiederholt, bis die kleinste noch nicht betrachtete unmarkierte Zahl größer als $\sqrt{n}$ ist. Die noch nicht markierten Zahlen sind dann die gesuchten Primzahlen.

In der Datei `eratosthenes25_animation.pdf` finden Sie eine Visualisierung des Algorithmus für $n = 25$, in der die 25 Zahlen in einem Quadrat angeordnet werden. Das Programm zur Darstellung der PDF-Datei muss allerdings mit „Overlays“ umgehen können. Das gilt z.B. für den Adobe Acrobat Reader ™. Falls kein geeigneter PDF-Viewer zur Verfügung steht, schauen Sie sich die statische Darstellung in `eratosthenes25_steps.pdf` an.

Die Markierung lässt sich am besten mit einer Liste, nennen wir sie `marked`, von boolschen Werten (d.h. jeder Wert der Liste ist entweder `True` oder `False`) repräsentieren: Der i-te Wert in der Liste ist genau dann `True`, wenn die Zahl i markiert ist. Am Anfang müssen alle Werte ab Index 2 auf `False` gesetzt werden, da noch keine Zahl markiert ist. Die Werte an Index 0 und 1 in der Liste können Sie auf `None` setzen, da sie nicht benötigt werden. Die einzelnen Phasen des Algorithmus bestehen darin, zu testen, ob eine Zahl `i` markiert ist (dann gilt `marked[i]`) bzw. nicht markierte Zahlen zu markieren, d.h. `marked[i] = True` zu setzen, entsprechend der obigen Beschreibung des Algorithmus.

Benennen Sie die Datei `eratosthenes_template.py` in `eratosthenes.py` um. Vervollständigen Sie die Funktion `prime_list_by_eratosthenes`, die den beschriebenen Algorithmus implementiert und die Liste der Primzahlen entsprechend des übergebenen Wertes von n zurückliefert.

Es ist bereits Programmcode zum Extrahieren einer ganzen Zahl n aus `sys.argv[1]` vorhanden. Es wird die Anzahl der Primzahlen $\leq n$ sowie die Liste der 10 größten Primzahlen $\leq n$ ausgegeben. Im Material zu dieser Übungsaufgabe finden Sie mehrere Dateien mit der erwarteten Ausgabe, je nach Wahl von n. Durch Aufruf von `make test` verifizieren Sie die Korrektheit Ihrer Implementierung für die verschiedenen Testfälle.

Punkteverteilung:

- 1 Punkt für das korrekte Setzen der Markierungen.
- 1 Punkt für die korrekte Bestimmung des nächsten unmarkierten Zeichens.
- 1 Punkt für die korrekte Berechnung der Liste der Primzahlen aus der Liste `marked` (inklusive bestandenen Tests).

Aufgabe 7.4 (4 Punkte)

In Textverarbeitungssystemen und auf Webseiten wird Text in Paragraphen häufig dynamisch umgebrochen. Wenn man mit Paste/Copy aus solchen Kontexten Text kopiert und in den Editor einfügt, entstehen meist sehr lange Zeilen, die jeweils den Inhalt eines Paragraphen darstellen. Ein Beispiel finden Sie in der Datei `python_history.txt` in den Materialien zu dieser Aufgabe.

Implementieren Sie eine Funktion `fold_text`, die den Inhalt einer Datei zeilenweise prozessiert und die Zeilen durch Einfügen des newline-Zeichens \n aufbricht und mit `print` ausgibt. Dabei müssen jeweils möglichst viele Worte (d.h. Strings ohne Leerzeichen) in einer Zeile stehen. Die Anzahl der Zeichen pro Zeile (ohne das newline-Symbol) soll nicht größer sein als ein vorgegebener Wert `max_line_width` (erster Parameter der Funktion). Das Umbrechen darf nur an Stellen erfolgen, an denen ein Leerzeichen steht oder nach dem letzten Wort einer eingelesenen Zeile. Die ausgegebenen Zeilen dürfen nicht mit einem Leerzeichen enden.

Damit Sie sich auf das Wesentliche konzentrieren können, finden Sie in den Materialien eine Datei `fold_text_template.py`. Benennen Sie die Datei in `fold_text.py` um. Der Kopf der Funktion `fold_text`, in der die Datei zum Lesen geöffnet und der geöffnete Stream wieder geschlossen wird, und einige weitere Zeilen existieren bereits in der Vorlage. Ebenso finden Sie Programmtext zum Überprüfen der Argumente aus `sys.argv`.

Beim Aufruf des Programms `fold_text.py` ist das erste Argument der Wert von `max_line_width` (also die maximale Anzahl der Zeichen pro Zeile in der Ausgabe) und das zweite Argument ist der Dateiname. Beispiel: Durch den Aufruf

```
./fold_text.py 70 python_history.txt
```

soll der Inhalt der Datei `python_history_fold70.txt` im Terminal ausgegeben werden. In dieser Datei haben alle Zeilen eine maximale Länge von 70 Zeichen.

Hinweis: Sammeln Sie die Worte, die zu einer Zeile gehören, zunächst in einer Liste. Sorgen Sie dafür, dass die Zeile, die aus dieser Liste entsteht, möglichst viele Worte enthält, aber niemals länger wird als `max_line_width`. Die Liste wird jeweils mit `join` und einem passenden Separator als String mit `print` ausgegeben.

Durch `make test` verifizieren Sie die Korrektheit Ihres Programms für einige Testfälle.

Punkteverteilung:

- 3 Punkte für eine strukturierte Implementierung von `fold_text`.
- 1 Punkt für die bestandenen Tests.

Bitte die Lösungen zu diesen Aufgaben bis zum 12.12.2022 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. Eine Studierender des Jahrgangs 2021 hat gefragt, warum das so ist und was genau Effizienz in diesem Zusammenhang bedeutet.

Ich will hier kurze Antworten geben. Das Thema Effizienz von Algorithmen wird im Modul Algorithmen und Datenstrukturen genauer 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 z.B. Faktoren linearer Terme 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. Ihre Anzahl hängt nicht von der Anzahl der Schlüssel im Dictionary ab.

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 ermitteln.