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

Programmierung für Naturwissenschaften 1
Wintersemester 2022/2023
Übungen zur Vorlesung: Ausgabe am 30.11.2022

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

Um ihre eigenen Fähigkeiten in der Python-Programmierung zu testen, lösen Sie bitte selbständig die folgenden Aufgaben. Nutzen Sie zunächst 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. Die mit ∗∗ gekennzeichneten Aufgaben sind etwas schwieriger.

1. ∗∗ Sei `n` eine Variable, die eine positive ganze Zahl speichert. Bestimmen Sie, in Abhängigkeit von `n`, die Anzahl der Aufrufe der Funktion `print` im folgenden Programmstück. Gesucht ist eine möglichst einfacher Ausdruck, der es erlaubt die Anzahl zu berechnen.

   ```
   for  i in range(1,n+1):
     for j in range(i+1,n+1):
       print('{}\t{}'.format(i,j))
   ```

2. Eine E-mail Adresse besteht aus dem Namen der Empfängerin, dem Symbol `@`, dem Namen der Domain, und der Top-Level Domain. Die Bestandteile kommen genau in dieser Reihenfolge vor. Die letzten beiden Bestandteile werden durch einen Punkt getrennt. So ist in der E-mail Adresse `jane.doe@icloud.com`, `jane.doe` der Name des Empfängerin, `icloud` ist der Name der Domain und `com` ist die Top-Level Domain. Alle genannten Bestandteile einer E-mail enthalten mindestens ein Zeichen. Der Name der Empfängerin sowie die Domain darf jeweils Groß- und Kleinbuchstaben sowie Ziffern und die Zeichen − (Bindestrich) und . (Punkt) enthalten. Die Top-Level Domain darf nur Buchstaben und Ziffern enthalten. Geben Sie einen regulären Ausdruck an, der spezifisch auf E-mail Adressen entsprechend dieser Definition passt. Beachten Sie, dass E-mail Adressen tatsächlich noch andere Sonderzeichen enthalten können und es Einschränkungen bei der Benutzung von Sonderzeichen gibt. Diese sollen hier aber nicht berücksichtigt werden.

3. Nehmen wir an, in einer Datei finden wir Sequenzen in beliebig vielen Zeilen, die jeweils mit einer positiven ganzen Zahlen beginnen, wie im folgenden Beispiel:

   ```
     1 aagcccagca gccccggggc ggatggctcc ggccgcctgg
    41 cgccctcctg cccccgatgc tgctgctgct gctccagccg
   ```

   Dabei gibt die Zahl die Position des nächsten Zeichens in der Sequenz an. Sei `sequence_lines` die Liste der Zeilen in Form von Strings, wie sie durch die Funktion `readlines()` geliefert wird. Geben Sie Programmzeilen in Python3 an, durch die aus `sequence_lines` die Sequenz als String berechnet werden kann. Diesert String besteht nur aus den Zeichen `a`, `c`, `g` und `t`.

4. Geben Sie bitte an, was das folgende Programmstück ausgibt. Begründen Sie Ihre Antwort.

```
lines_list = ['Im Fruehling bluehen die Blumen.',
              'Der Fruehlingsanfang ist immer im Maerz',
              'Der Sommer war in diesem Jahr sehr heiss.',
              'Am Herbstende ist Winterwetter normal.']
jahreszeiten = ['Fruehling','Sommer','Herbst','Winter']
re_jahreszeiten = '|'.join(jahreszeiten)
for line in lines_list:
  line = line.rstrip()
  if re.search(r'\b({})\b'.format(re_jahreszeiten),line):
    print(line)
```

5. Geben Sie bitte an, was das folgende Programmstück ausgibt. Welche bekannte Matrixoperation wird hier implementiert?

```
matrix0 = [[0,1,3],\
           [4,5,6],\
           [7,8,9]]
matrix1 = list()
for i in range(len(matrix0)):
  new_row = list()
  for row in matrix0:
    new_row.append(row[i])
  matrix1.append(new_row)
print(matrix1)
```

6. Nehmen wir an, dass die Datei mit dem Namen `simple.fna` 10 Zeilen enthält. Korrigieren Sie den logischen Fehler im folgenden Programm.

```
try:
  stream = open('simple.fna')
except IOError as err:
  sys.stderr.write('{}: {}\n'.format(sys.argv[0],err))
  exit(1)
max_len = 0
for line in stream:
  if max_len < len(line):
    max_len = len(line)
  stream.close()
```

7. Finden Sie den Fehler im folgenden Programm und korrigieren Sie diesen.

```
import sys
op_sys = ['SUSE-linux','ubuntu-linux','macOS','Red hat Linux']
if 'x86/MS-Windows' not in op_sys:
  sys.stderr.write('{}: windows not available, choose one of\n')
  for sys in op_sys:
    sys.stderr.write('{}\n'.format(sys))
```

Aufgabe 6.1 (4 Punkte) Sie sind als studentische Hilfskraft im Eimsbütteler Chronik-Verlag beschäftigt. Dieser Verlag verkauft Chroniken verschiedener Länder. Für die nächste Auflage der Chroniken sollen die Datumsangaben in einem neuen vereinheitlichten Format erscheinen. Das bisherige Format besteht aus einer Tageszahl (ein oder zweistellig), einem Monatsbezeichner (Januar, Februar, ..., November, Dezember) und einer vierstelligen Jahreszahl. Jedes Datum in diesem Format soll durch das entsprechende Datum im Format `tt.mm.yyyy` ersetzt werden. Sonst soll in den Chroniken nichts verändert werden. Sie können davon ausgehen, das alle Angaben zu einem Datum immer in der gleichen Zeile stehen. Um die Qualität Ihrer Arbeit zu verifizieren, hat man einen Testdatensatz erstellt. Dieser besteht aus einer Datei mit einer Chronik, die 85 Datumsangaben enthält und einer weiteren Datei mit der gleichen Chronik, in der Ersetzung der Datumsangaben bereits erfolgt ist.

Entwickeln Sie ein Python-Programm `convert_dates.py`, das genau einen Dateinamen über `sys.argv` erwartet. Wenn das Skript nicht mit genau einem Argument aufgerufen wird oder die Datei nicht geöffnet werden kann, geben Sie entsprechende Fehlermeldungen aus und brechen das Programm mit `exit(1)` ab.

Das Programm soll die Datei, deren Name übergeben wurde, zeilenweise einlesen und verarbeiten. Es soll zu jedem Zeitpunkt immer nur höchstens eine Originalzeile und eine modifizierte Zeile gespeichert werden. Nach der Ersetzung aller Datumsangaben in einer Zeile unter Verwendung eines Dictionaries für die Monatskonvertierung wird die modifizierte Zeile mit `print` ausgegeben. Für einige von Ihnen wird vermutlich die Aufgabenbeschreibung jetzt schon ausreichen, um das Programm erfolgreich entwickeln zu können. Wenn Sie sich zu dieser Gruppe von Studierenden zählen, können Sie die folgenden Hinweise überspringen.

Hinweise zur Entwicklung des Programms

Für die Konvertierung eines Monatsbezeichners in die entsprechende Monatsnummer berechnen Sie ein Dictionary `month2number`. Die Schlüssel hiervon sind die Monatsbezeichner und die Werte sind die entsprechenden Monatsnummern. In einem Programm einer früheren Aufgabe finden Sie bereits eine Liste mit den Monatsbezeichnern, die Sie hier zur Berechnung von `month2number` wiederverwenden müssen.

Verarbeiten Sie jede einzelne Zeile der Eingabedaten (jeweils nach Löschen des Zeichens `\n` am Ende) in zwei Schritten wie folgt:

Extraktion: Berechnen Sie zunächst mit `re.findall` und einem regulären Ausdruck alle Datumsangaben im Originalformat. Benutzen Sie in Ihrem regulären Ausdruck Gruppenklammern, damit Sie die einzelnen Bestandteile eines Datums extrahieren können. Klammern Sie auch den gesamten RE, damit Sie einfach auf die gesamten matchenden String zugreifen können. Beachten Sie, dass `re.findall` für jeden Treffer eine Liste der Strings, die zu den Gruppen in Ihrem regulären Ausdruck passen, liefert. Sie müssen selbst herausfinden, an welcher Position der Liste welcher String steht. Berechnen Sie aus dieser Liste den String mit dem neuen Datumsformat. Erzeugen Sie für jede Zeile ein Dictionary `date_pair_dict`, dessen Schlüssel eine Datumsangabe im Originalformat ist und dessen Wert das Datum im neuen Format ist. Beispiel: Für die Zeile

```
1. Oktober 1948 bis zum 30.September 1949 rund 4.491,5
```

ergibt sich das folgende Dictionary.

```
{'1. Oktober 1948' : '01.10.1948', '30.September 1949' : '30.09.1949'}
```

Beachten Sie, dass das Dictionary für jede Zeile der Eingabedatei initialisiert und berechnet wird und dass das Dictionary für $1013 - 78 = 935$ Zeilen der Testdatei leer ist.

Ersetzung: Verwenden Sie `re.sub`, um für jedes Schlüssel/Wert-Paar des zuletzt genannten Dictionaries den Schlüssel durch den Wert zu ersetzen. Nachdem alle Ersetzungen durchgeführt wurden, geben sie die modifizierte Zeile mit `print` aus.

Punkteverteilung

- 1 Punkt: Berechnung des Dictionaries für die Monatskonvertierung
- 2 Punkte: prinzipiell funktionierende Konvertierung der Zeilen
- 1 Punkt: bestandene Tests

Aufgabe 6.2 (7 Punkte)

In dieser Aufgabe geht es darum, aus einer Textdatei ganze Zahlen und Fließkommazahlen zu extrahieren und hierfür jeweils eine Verteilung zu berechnen. Der Inhalt der Textdatei wurde von Blast, einem Programm zum Vergleich biologischer Sequenzen, ausgegeben und für diese Aufgabe vereinfacht. Uns interessieren hier drei Werte aus den Ergebnissen des Sequenzvergleichs, nämlich der Bitscore (positive ganze Zahl), der Erwartungswert (Expect, 0.0 oder reelle Zahl in wissenschaftlicher Notation) und der Anteil der identischen Zeichen (Identities, positive ganze Zahl). Was diese genau bedeuten, ist in dieser Aufgabe nicht wichtig. Ihr Programm soll Zeilen der Blast-Ausgabe, die wie folgt aussehen, erkennen und daraus Werte extrahieren:

```
Score = 132 bits (332), Expect = 5e-32
Identities = 108/341 (32%)
Score = 46 bits (107), Expect = 6e-05
Identities = 26/62 (42%)
```

Aus den ersten beiden Zeilen sollen die Werte `132` (der Bitscore), `5e-32` (der Erwartungswert) und `32` (der relative Anteil der identischen Zeichen beim Sequenzvergleich) extrahiert werden. Aus den letzten beiden Zeilen sind das die Werte `46`, `6e-05` und `42`.

Es sollen aus allen Zeilen der Eingabedatei (wenn sie die syntaktischen Bedingungen erfüllen) die Werte extrahiert und drei diskrete Verteilungen[1] ausgegeben werden, und zwar jeweils eine für die drei genannten Werte. Gehen Sie wie folgt vor:

- Geben Sie für jede Zeile aus der Eingabedatei, in der ein oder zwei Werte erkannt wurden, diese Werte in einer Zeile der Form `V` *b e* bzw. `V` *i* aus. Dabei ist *b* der Bitscore, *e* der gerundete 10er Logarithmus des Erwartungswerts und *i* der Identitätswert aus der entsprechenden Zeile. Das Zeichen `V` und die Werte sind jeweils durch einen Tabulator getrennt. Beispiel: Aus den oben gezeigten Zeilen ergibt sich damit die folgende Ausgabe:

  ```
  V 132 -31
  V 32
  V 46 -4
  V 42
  ```

[1] Ein diskrete Verteilung ist das Ergebnis einer Zählung. Sie gibt also an, wie häufig etwas vorkommt, z.B. die Anzahl der Vorkommen der Score-Werte.

Beachten Sie, dass $\log_{10}(\texttt{6e}-\texttt{05}) \approx -4.2218$. Daher steht der gerundete Wert `-4` in der dritten Zeile.

- Nach den V-Zeilen soll als Erstes die Verteilung der Bitscores ausgegeben werden.
- Als Nächstes soll die Verteilung der 10er-Logarithmen der Erwartungswerte ausgegeben werden. Zur Berechnung der log-Werte verwenden Sie die Funktion `log10` aus dem Math-Modul. Dazu muss die Zeile `from math import log10` am Anfang Ihres Python-Skriptes eingefügt werden. Die Rundung der log-Werte erfolgt mit der Funktion `round`. Da `log10(0)` nicht definiert ist, müssen Sie den Erwartungswert 0.0 gesondert behandeln und als Exponent 0 zurückliefern (auch wenn das mathematisch nicht korrekt ist).
- Als Letztes soll die Verteilung der Prozent-Identitäts-Werte ausgegeben werden.

Das Format der Ausgabe für die Verteilungen ist unten angegeben.

Die einzelnen Werte und Strings in den Zeilen der Eingabe sind jeweils durch beliebig viele Leerzeichen getrennt. Verwenden Sie zwei verschiedene reguläre Ausdrücke mit Gruppen (ausgedrückt durch Paare von runden Klammern), um die genannten Werte zu extrahieren. Die regulären Ausdrücke sollen spezifisch für die beiden Zeilenformen sein, d.h. sie müssen auch die Schlüsselworte `Score`, `Expect`, `Identities` und `bits` sowie die Zeichen `=`, `,` und `%` berücksichtigen. Ebenso müssen die anderen Werte, die für die Weiterverarbeitung nicht von Interesse sind, durch die REs gematcht werden. Zur systematischen Entwicklung der REs können Sie z.B. das Web-Tool `https://pythex.org` verwenden.

Falls einer der regulären Ausdrücke matcht, liefert die Methode `group` die extrahierten Werte als Strings. Diese müssen für die Weiterverarbeitung mit den Methoden `int` und/oder `float` konvertiert werden.

Um die Verteilungen zu speichern, verwenden Sie Dictionaries. Denken Sie daran, für Schlüssel, die bisher noch nicht im Dictionary vorkommen, den richtigen Initialwert zu verwenden. Die Verteilungen sollen Tabulator-separiert und numerisch sortiert nach den Schlüsseln der Dictionaries ausgegeben werden. Dazu können Sie Programmzeilen der Form

```
for k in sorted(dist):
    print('D\t{}\t{}'.format(k,dist[k]))
```

verwenden. Dabei ist `dist` das dictionary, das die Verteilung speichert.

Es gibt 3 Tests und 4 Testdateien, um die Korrektheit Ihrer Implementierung zu verifizieren. Die Eingabedatei für die ersten beiden Tests ist `blaststat.txt` mit insgesamt 100 Zeilen der obigen Form.

- Durch `make test_values` wird verifiziert, dass die `V`-Zeilen die richtigen Werte enthalten, nämlich genau die, die in der Datei `blaststat_values.tsv` stehen. Erst wenn dieser Test erfolgreich bestanden ist, implementieren Sie die Schritte zur Berechnung der Verteilungen. Die `V`-Zeilen erlauben Ihnen eine systematische Fehlersuche in Ihrem Programm.
- Durch `make test_distrib` wird verifiziert, dass die ausgegebenen Verteilungen die richtigen Werte enthalten, nämlich genau die, die in der Datei `blaststat_distrib.tsv` stehen. Beachten Sie, dass letztere Datei zwei Kopfzeilen enthält, die Ihr Programm mit ausgeben muss, damit der Test bestanden wird.
- Durch `make test_corrupt` wird verifiziert, dass Ihre regulären Ausdrücke spezifisch genug sind. Das erfolgt mit der Datei `blaststat_corrupt.txt`, die fehlerhafte Einträge

mit Score und Identities-Werten enthält, aus denen sich eine leere Verteilung ergibt. Nach dem Prozessieren aller Zeilen dieser Datei liefert Ihr Programm entsprechend eine leere Ausgabe, denn bei einer leeren Verteilung sollen auch die Kopfzeilen nicht mit ausgegeben werden.

Benennen Sie die Datei `blaststat_template.py` aus den Materialien in `blaststat.py` um. In dieser Datei erfolgt Ihre Implementierung. Nachdem Sie alle Teile Ihres Programms, wie oben beschrieben, getestet haben, verifizieren Sie durch `make test` nochmals die Korrektheit Ihres Programms für alle Testdaten.

Hinweise zur Lösung: `python-slides.pdf`, Abschnitt *Regular expressions*, frame 1-11

Hinweise zur Lösung: `python-slides.pdf`, Abschnitt *Histograms: counting occurrences of values*, frame 17-19

Punkteverteilung:

- jeweils 2 Punkte für die beiden korrekten regulären Ausdrücke,
- 1 Punkt für die korrekte Extraktion der Werte und den bestandenen Test `test_values`
- 1 Punkte für die korrekten Initialisierungen und Aktualisierungen der Dictionaries und den bestandenen Test `test_distrib`
- 1 Punkt für den bestandenen Test `test_corrupt`.

Bitte die Lösungen zu diesen Aufgaben bis zum 05.12.2022 um 18:00 Uhr an pfn1@zbh.uni-hamburg.de schicken.