

Programmierung für Naturwissenschaften 1
Wintersemester 2020/2020
Übungen zur Vorlesung: Ausgabe am 09.12.2020

Aufgabe 5.1 (3 Punkte) Schreiben Sie ein Programm `datetonenumber.py`, das genau ein Argument von der Kommandozeile erhält, und zwar den Namen einer Datei. Diese Datei soll in jeder Zeile ausschließlich ein Datum in der Form `DD.MM.JJJJ` enthalten. Ihr Programm soll eine Datei in einem solchen Format einlesen und zu jedem Datum nach einem Tabulatorzeichen die Nummer des Tages im gesamten Jahr angeben.

Beispiel: Nehmen wir an, die Datei `randomdates.csv` enthält folgende Zeilen:

```
05.03.2017
27.09.2006
09.11.2010
24.05.2011
17.11.2000
```

Dann soll `./datetonenumber.py randomdates.csv` die folgende Ausgabe liefern. Dabei steht zwischen dem Datum und der Nummer des Tages jeweils ein Tabulatorzeichen.

```
05.03.2017 64
27.09.2006 270
09.11.2010 313
24.05.2011 144
17.11.2000 322
```

Bitte beachten Sie Schaltjahre. Um zu ermitteln, ob ein Jahr ein Schaltjahr ist, können Sie Teile der Lösung einer früheren Aufgabe wiederverwenden.

In den Materialien zur Übung finden Sie eine Testdatei und ein `Makefile`. Darin ist ein Test implementiert, der Ihr Programm aufruft und die Ausgabe mit dem erwarteten Ergebnis vergleicht. Zudem wird überprüft, dass Ihr Programm bei fehlerhaften Aufrufen die richtige Fehlermeldung ausgibt und mit `exit(1)` abbricht. Durch `make test` verifizieren Sie die Korrektheit Ihres Programms.

Hinweise zur Lösung: `python-slides.pdf`, Abschnitt *Flow of control*, frame 37

Hinweise zur Lösung: `python-slides.pdf`, Abschnitt *Regular expressions*, frame 5

Punkteverteilung:

- 1 Punkt für die korrekte Behandlung mit zu wenigen Parametern oder Dateien, die sich nicht öffnen lassen.
- 1 Punkt: korrekte Extraktion der Werte für das Jahr, den Monat und den Tag
- 1 Punkt für die Berechnung der Nummer des Tages (inklusive Tests)

Aufgabe 52 (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 wurde für die Aufgabe vereinfacht. Uns interessieren hier drei Schlüsselwerte 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 Verteilungen (Histogramme) berechnet werden, jeweils eine für die drei genannten Werte.

- Es soll für jede Zeile aus der Eingabedatei, in der ein oder zwei Werte erkannt wurden, diese Werte in einer Zeile der Form $\vee b e$ bzw. $\vee i$ ausgegeben werden, wobei b der Bitscore, e der gerundete 10er Logarithmus des Erwartungswerts und i der Identitätswert aus der entsprechenden Zeile ist. Das Zeichen \vee und die Werte sind jeweils durch einen Tabulator getrennt.
- Es soll die Verteilung der Werte der Bitscores ausgegeben werden.
- Es 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. Den Fall, dass der Erwartungswert 0 ist, müssen Sie gesondert behandeln, denn `log10(0)` ist undefiniert.
- Es 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 *Captures* (ausgedrückt durch Paare von runden Klammern), um die genannten Werte zu extrahieren. Die regulären Ausdrücke sollen spezifisch für die beiden Zeilenformen sein, d.h. auch die Schlüsselworte `Score`, `Expect` und `Identities` und die Teile der Zeilen berücksichtigen, die Werte enthalten, die hier nicht von Interesse sind. 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 noch nicht im Dictionary vorkommen, den Initialwert 0 zu speichern. Die Verteilungen sollen

Tabulator-separiert und numerisch sortiert nach den Schlüsselwerten 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.
- 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 funktioniert.
- Durch `make test_corrupt` wird verifiziert, dass keine der Zeilen in `blaststat_corrupt.txt` entsprechend des obigen Musters formatiert sind. 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 14.12.2020 um 18:00 Uhr an `pfn1@zbh.uni-hamburg.de` schicken.