Informatik I: Einführung in die Programmierung

Prof. Dr. Bernhard Nebel
Dr. Christian Becker-Asano, Dr. Stefan Wölfl
Wintersemester 2013/2014

Universität Freiburg
Institut für Informatik

Übungsblatt 9

Abgabe: Freitag, 17. Dezember 2013, 18:00 Uhr

Bei diesem Übungsblatt erwarten wir alle Lösungen zu den Aufgaben in der jeweils angegebenen Datei im Unterverzeichnis `sheet09`. Beachten Sie dabei die bekannten Formatierungshinweise für Python-Dateien und vergessen Sie nicht, alle von Ihnen verwendeten Dateien ins SVN-Repository hochzuladen!

Aufgabe 9.1 (Reguläre Ausdrücke; Punkte: 3+3; Datei: `ex09-1.txt`)

Beschreiben Sie umgangssprachlich, welche Mengen von Strings von den folgenden regulären Ausdrücken spezifiziert werden. Geben Sie dann für jeden regulären Ausdruck jeweils zwei Strings an, die den regulären Ausdruck matchen, und zwei, die den Ausdruck nicht matchen.

(a) `r'(?<![+?]{2})([^+]+?)\++'`

(b) `r'\b(([A-Za-z]+?)\2)\B'`

Aufgabe 9.2 (Reguläre Ausdrücke II; Punkte: 3+3; Datei: `re_examples.py`)

(a) Schreiben Sie eine Funktion `replace_brackets`, die in einem gegebenen String für jede mit runden Klammern eingeklammerte Folge von `\w`- oder Weißraum-Zeichen (wie zum Beispiel diese Bemerkung) die runden Klammern durch entsprechende eckige Klammern ersetzt (also `'('` durch `'['` und `')'` durch `']'`) und diesen String zurückgibt.

Hinweis: Die Methode `re.sub` unterstützt auch Rückreferenzen auf Gruppen.

(b) Schreiben Sie eine Funktion `find_dates`, die aus einem Text alle Datumsangaben der Gestalt wie im Beispiel

```
10. März 2014, 9:05 Uhr
```

findet und als eine Liste von Dictionaries zurückgibt. Jedes Dictionary habe dabei die Schlüssel `year`, `month`, `day`, `hour`, und `minute`, denen numerische Werte zugewiesen sind. Das Dictionary im Beispiel würde also wie folgt aussehen:

```
{'year': 2014, 'month': 3, 'day': 10, 'hour': 9,
'minute': 5}.
```

Testen Sie Ihre Funktionen an jeweils 3 Beispielen als Doc-Tests der jeweiligen Funktion.

Aufgabe 9.3 (Reguläre Ausdrücke und Gen-Sequenzierung; Punkte: 1+1+4; Datei: `augt.py`)

In der 20. Vorlesung wurde beispielhaft ein regulärer Ausdruck zum Extrahieren von URLs und Linktexten vorgestellt. In Anlehnung an dieses Beispiel, wollen wir uns mit einem Problem beschäftigen, das für die Bioinformatik relevant ist.

Der genetische Code eines jeden (irdischen) Lebewesens beschreibt eine Regel, nach der Sequenzen aus Nukleobasen, die durch die vier Zeichen `AUGT` repräsentiert werden können, in Aminosäuren übersetzt werden[1]. Auf spezifische Details wollen wir nicht weiter eingehen, sondern uns im Weiteren auf ein Teilproblem dieses Prozesses konzentrieren.

Beliebig lange Strings aus den Zeichen `AUGT` sind prinzipiell möglich. Jede beliebige Dreiergruppe innerhalb dieses Strings wird als *Codon* bezeichnet, womit $4^3 = 64$ mögliche Basentripletts/Codons möglich sind. Folgende (stark vereinfachte) Regeln bei der Transkription einer Sequenz sind zu beachten:

- Die Übersetzung einer Nukleobasen-Sequenz kann an der ersten, zweiten oder dritten Stelle begonnen werden, wodurch sich verschiedene `Codon`-Sequenzen ergeben.
- Bei einmal festgelegtem Startpunkt werden alle weiteren Nukleobasen lückenlos zu Codons zusammengefasst und übersetzt.
- Das Codon `AUG` wird als *Startcodon* bezeichnet.
- Das Codon `UAG` wird als *Stopcodon* bezeichnet.
- Alle Codons zwischen einem Start- und einem Stopcodon sind (für uns) von Bedeutung, alles andere sollte ignoriert werden.

Nun zu Ihren Programmieraufgaben, bei denen Sie davon ausgehen dürfen, dass alle Eingabestrings keine anderen als die Zeichen `AUGT` (beliebig oft) enthalten:

(a) Implementieren Sie eine Funktion `check_for_startcodon(AUGTstring)`, die prüft, ob der Eingabestring `AUGTstring` mind. ein Startcodon enthält und nur in diesem Fall `True` zurückliefert, sonst `False`. Ist es hierfür sinnvoll, reguläre Ausdrücke zu verwenden? Begründen Sie.

(b) Entwerfen Sie einen regulären Ausdruck, mit dem geprüft wird, ob irgendwo innerhalb eines Eingabestrings mind. einmal auf ein Startcodon `AUG` ein Stopcodon `UAG` folgt. Hierbei darf alles zwischen diesen beiden Codons ignoriert werden.

[1] Siehe auch: `http://de.wikipedia.org/wiki/Genetischer_Code`

(c) Schreiben Sie unter Benutzung des Moduls `re` für reguläre Ausdrücke eine Funktion `extract_codon_string(AUGTstring)`, die folgende Anforderungen erfüllt:

- Alle Codons, die durch ein Start- und ein Stopcodon eingeschlossen werden, sollen aus dem Eingabestring `AUGTstring` herausgefiltert und als Liste zurückgegeben werden.
- Der Rückgabewert der Funktion ist also eine Liste von Strings, deren jeweilge Länge ein ganzes Vielfaches von drei sein muss. Diese Bedingung sollte als Teil des verwendeten regulären Ausdrucks implementiert werden.
- Ihr regulärer Ausdruck sollte `non-greedy` sein. Zum Beispiel sollte der Eingabestring `'AUGTTTUAGAUGTTTUAG'` die Liste `['TTT', 'TTT']` als Rückgabewert liefern.
- Eingebettete `AUG`'s sind erlaubt, d.h. die Eingabe `'AUGAUGTTTUAG'` sollte `['AUGTTT']` als Rückgabewert liefern.

Dokumentieren und testen Sie Ihre Implementationen wie bei den vorherigen Übungsblättern durch Angabe geeigneter Doc-Strings.

Aufgabe 9.4 (Erfahrungen; 2 Punkte)

Legen Sie im Unterverzeichnis `sheet09` eine Textdatei `erfahrung.txt` an. Notieren Sie in dieser Datei kurz Ihre Erfahrungen beim Bearbeiten der Übungsaufgaben (Probleme, benötigter Zeitaufwand nach Teilaufgabe, Bezug zur Vorlesung, Interessantes, etc.).