Übung 01: Input/Output, Encoding, Classes

Programmiertechniken in der Computerlinguistik II, FS 16

Abgabedatum: 17. März 2016, 18:00 Uhr

Hinweise zur Abgabe

- Bitte gib jedes Python-Programm in einer eigenen Datei ab, die die Dateiendung `.py` hat und *ausführbaren* Python-Code enthält.
- Geize nicht mit Kommentaren direkt im Programm-Code, wo Erläuterungen angebracht sind. Umfangreiche Erklärungen werden hingegen besser in einer separaten README-Datei mitgeliefert (vorzugsweise Plain-Text oder PDF).
- Jedes Python-Skript sollte korrekt formatiert und importierbar sein, ohne dass Code ausgeführt wird.
- Um das Hochladen der Abgabe auf OLAT zu erleichtern, kannst du die Dateien mit `zip` oder `tar` (oder einem anderen verbreiteten Format) archvieren / komprimieren.

1 Coding-Stil und Programmierprinzipien

1.1

Lies dir den offiziellen Python Style-Guide (https://www.python.org/dev/peps/pep-0008/) und die Google-Ergänzungen dazu (https://google.github.io/styleguide/pyguide.html#Python_Style_Rules) durch. Versuche, deinen Code wenn möglich danach zu gestalten. Alle Übungen in PCL II werden auch nach Coding-Stil bewertet. Du musst nicht unbedingt in allen Einzelheiten dem PEP08 folgen, aber es ist ein guter Ansatzpunkt.

1.2

Für die Lesbarkeit eines Programms ist nicht nur das Layout wichtig, sondern auch die Organisation des Codes. Ein wichtiges Prinzip ist das Prinzip der Modularität, also die Aufteilung eines Programms in kleinere, wohldefinierte Bestandteile. Es gibt aber auch andere solche "Prinzipien der Programmierung". Recherchiere einige dieser Prinzipien im Internet (zum Beispiel auf Wikipedia: https://en.wikipedia.org/wiki/Category:Programming_principles) und notiere drei, die dir besonders einleuchten. Weshalb sind sie wichtig?

1.3

Nimm ein altes Python-Skript von dir, zum Beispiel aus PCL I, und kritisiere es. Welche Stil- und Organisationsprobleme findest du? Das Skript sollte ungefähr 50-80 Zeilen lang sein. (Wenn du kein geeignetes Skript hast, darfst du auch eins von jemand anderem nehmen.)

2 Fortgeschrittene Klassen

Wir möchten in Python einen Paket-Manager für die Linux-Distribution Y schreiben. Die Distribution Y ist aus dem Zusammenschluss zweier kleinerer Distributionen entstanden. Beide dieser Distributionen hatten bis anhin einen eigenen Paket-Manager, die beide ein anderes Format für Paket-Konfigurationsdateien verwendeten, weshalb die Paketdatenbanken für Y Linux ein wildes Gemisch beider Formate sind.

Glücklicherweise werden beide Formate von der Python-Standardbibliothek unterstützt: die Hälfte der Dateien ist im JSON-Format, die andere im .ini-Format.

Um die Architektur unseres Systems eleganter zu gestalten, möchten wir gerne auf die Konfigurationsdateien formatunabhängig zugreifen können. Zu diesem Zweck sollst du eine Klasse `ConfigReader` schreiben, die ein solches Interface implementiert.

Deine Klasse soll beim Aufruf einen Dateinamen erhalten und das Format erkennen. Um die beiden unterschiedlichen Formate zu bearbeiten, sollst du zwei Subklassen definieren: `JsonReader` und `IniReader`. Diese Subklassen sollen Dateien des jeweiligen Formats einlesen und sich nach aussen wie Dictionaries der Schlüssel-Wert-Paare in der eingelesenen Datei verhalten (definiere dazu die Methode `__getitem__`). Die Oberklasse `ConfigReader` soll die Methode `__new__` so definieren, dass je nach Format der übergebenen Datei die richtige Subklasse zurückgegeben wird. Zum Einlesen der Dateien kannst du die Standardmodule `json` und `ConfigParser` verwenden.

Im Übungsordner ist ein Skript zum Testen deines Moduls `test_installer.py` enthalten.

3 Input, Output, Encoding

Das UNIX-Programm `sed` ist ein beliebtes Tool, um Text in einer Pipeline zu editieren. Die am häufigsten genutzte Funktion dieses Programms ist der Befehl `s/pattern/replace/g`, um Ersetzungen mit regulären Ausdrücken durchzuführen. Du sollt nun ein Python-Programm schreiben, dass einen solchen Filter implementiert.

Dein Programm soll von der Kommandozeile einen Befehl in der Form `s/pattern/replacement/g` einlesen. Den Text soll es standardmässig von der Standardeingabe lesen und den bearbeiteten Text auf der Standardausgabe wieder ausgeben.

Das Programm sollte folgende Kommandozeilenoptionen unterstützen:

a) `-e|--encoding`: Das Encoding der eingelesenen Datei. Dein Programm sollte mindestens die Encodings `ascii`, `latin-1` und `utf-8` unterstützen.

b) `-f|--file`: Datei, aus der der Rohtext gelesen werden soll, wenn er nicht von der Standardeingabe kommt.

c) `-o|--out`: Datei, in die der bearbeitete Text geschrieben werden soll, wenn er nicht auf der Kommandozeile ausgegeben werden soll.

Um Unicode-kompatible reguläre Ausdrücke zu unterstützen, solltest du ausserdem statt des Standardmoduls `re` die Bibliothek `regex` verwenden. Du kannst dieses Paket von der Kommandozeile mithilfe von `$ sudo pip install regex` installieren. (Sollten Installationsprobleme auftreten, wende dich zuerst an die Tutoren; wenn sie nicht gelöst werden können, ist es in Ordnung, stattdessen `re` zu verwenden.)

Reflexion/Feedback

a) Fasse deine Erkenntnisse und Lernfortschritte in zwei Sätzen zusammen.

b) Wie viel Zeit hast du in diese Übungen investiert?