Dynamic Languages - Übungen Blatt 2

Abgabedatum: 3. Mai 2010
Subversion URL: svn://wyvern.cs.uni-duesseldorf.de/dynlang08

Aufgabe 1

1. Write a function that takes a string and computes a nested dictionary from it. The goal is to find out which words follow another word in this string. The dictionaries keys are therefore the words appearing in the string. The values are again dictionaries that map the following word to a number of occurrences. The test function is given in `blatt2.py`.
2. Such a dictionary can be used to produce a random text that is similar to the text the dictionary was created from. To do this, start with a random word from the dictionary. Then pick the next work out of the words that followed the first word in the original text, with probabilities according to their occurrences.

Aufgabe 2

1. Change `mygettattr` from last lecture to add a way to add "virtual" attributes to types without changing the type itself, which are only seen when using `mygettattr`. These overrides can be added with a helper functions. The test functions are given in `blatt2.py`.
2. Write a metaclass that exposes the overriding behaviour in a nice way. All the attributes of the class are to be added to the base type of the type class. The test functions are given in `aufgaben/blatt2.py`.

Aufgabe 3

(3 Punkte)
Write a metaclass `OpenClass` that can be used to implement open classes. If a class `A` of that metaclass is created, it acts as a normal class and should be subclassable normally. However, when a subclass of the special name `__enhance__` of the class is created, the attributes in the subclass are added to `A` instead of actually making a new class. There is a test given for this in `blatt2.py`, however you probably have to write more.

Aufgabe 4

A possible implementation of the forward part of the Burrows-Wheeler transformation (Aufgabe 4 Blatt 1) is given in `blatt2.py`. However, it has the disadvantage that it builds in memory all shifted versions of the input string, which uses memory that is O(n**2) where n is the length of the input string. The point of this exercise is to rewrite this function to use memory that is only linear in the input string length. This can be done by writing a class whose instances have a references to the original string and an shift number that indicates how much the string is shifted. The class needs a __lt__ method that does comparisons between such instances correctly. Write tests that test the behaviour of this class. In addition, the existing tests for the forward transformation should still pass.