RUB

Übung 8: Module & OOP

Sommersemester 2017

Helena Wedig

RUHR UNIVERSITÄT BOCHUM RUB
Sprachwissenschaftliches Institut

Gliederung

- Wiederholung der Vorlesung
- Präsentation der Hausaufgaben
- Korrektur HA6 – Besprechung
- Aufgaben

Wiederholung der Vorlesung – Module

- Modul = Datei mit Dateiendung mit Endung .py
  - Dateiname des Moduls bei Import als Variablennamen
  - Dienen der Modularisierung des Programms
- Vorteile von Modulen
  - Code-Wiederverwendung
  - Code-Strukturierung
  - Namensräume
- Aufruf von Methoden und Variablen durch Modul.methode()/attribut

Wiederholung der Vorlesung – OOP

- Klasse = Gruppe von Objekten, die in bestimmten Eigenschaften übereinstimmen
  - Schablonen oder Schemata für die Erzeugung von Objekten
- Objekte = konkrete, in einem Programm verwendete Instanzen von Klassen
- Beziehung zwischen Klasse und Objekt = is-a-Relation

- Form einer Klasse
  - Klassendefinition: class Name(superclass, ...)
  - Geteilte Klassendaten: attr = wert
  - Klassen-Methoden: def method(self, ...)
  - Instanz-spezifische Daten: self.attr = value

- Wichtig:
  - Klassennamen werden groß geschrieben
  - Klassen können von Superklassen erben
  - Klassen-Attribute stehen im Code der Klasse ohne self
  - Instanz-Attribute stehen mit self

Wiederholung der Vorlesung – OOP

- Einsatz von Konstruktoren, um Attribute und Default-Werte bei der Instanziierung zu setzen
  - class C1:
    def init (self, who): self.name = who
- Destruktoren werden beim Löschen einer Instanz aufgerufen
- Eigenen Iterator für eigene Klassen durch def __iter___(self) und __next__(self)

- Vererbung:
  - Eine Instanz einer Klasse ist verlinkt mit ihrer Klasse
  - Klassen können mit Oberklassen verlinkt sein und deren Eigenschaften erben
  - Oberklassen werden im Header gelistet

Korrektur HA6 – Besprechung

	Allgemein	
Kommentare		10
Programm läuft mit allen möglichen Eingaben		5
Variablennamen konformer Code		5
Programmstruktur		5
	Aufgabenspezifisch	
Aufgabe 1		
	read_file	**5**
Textdatei wird geöffnet		1
with wird verwendet		1
wird eingelesen		1
wird geschlossen		1
Rückgabe eines Strings		1
	read_grammar	**10**
verarbeitet String mit der Grammatik		2
liest Regel ein und speichert sie in entsprechender Datenstruktur		5
erklärt die Wahl der Datenstruktur in min. einem Satz		1
Gibt Grammatikregeln zurück		2
	write_grammar	**12**
Dateiname wird abgespeichert		1
Grammatikregeln werden eingelesen		2
Ausgabedatei wird geöffnet		1
Ausgabe wird in Datei geschrieben		2
Form: 2 Tabulatoren zwischen der Regel und ihrer Wahrscheinlichkeit		5
Ausgabedatei wird geschlossen		1
	extract_rules	**10**
Kategorie der Mutter wird eingelesen		1
Grammatikregeln werden eingelesen		1
Grammatikregeln mit entsprechendem Mutterknoten werden aus der Datenstruktur ausgelesen		5
ausgewählte Grammatik wird ausgegeben		3

Korrektur HA6 – Besprechung

Aufgabe 2		
read_treebank	**26**	
String wird eingelesen		3
Klammerformat wird eingelesen		5
Frequenzen werden errechnet		8
Rekursion wird verwendet		5
Baum wird mit Frequenzen in geeigneter Datenstruktur abgespeichert		5
write_grammar_in_file	**5**	
write_grammar wird aufgerufen		5
Hauptprogramm	**7**	
Grammatik wird eingelesen		1
Regeln werden geparst		1
Regeln werden extrahiert		1
Subset wird ausgegeben		1
Baumbank wird eingelesen		1
Baum wird geparest		1
Grammatik wird ausgegeben		1
GESAMT		100

Aufgaben

- Schreiben Sie eine Klasse „dictionary“ in einem Modul dictionary.py
- Wenn dieses Programm importiert wird, meldet es sich mit einer Ausgabe auf dem Bildschirm
- Folgende Methoden sollen aufgerufen werden können:
  - give_def gibt die Definition eines ausgewählten Begriffs zurück
  - set_def ermöglicht es, neue Definitionen zu setzen
  - del_def ermöglicht es, Definitionen zu löschen
- Alle Ausgaben sollen dabei in einer Datei namens “verlauf.txt“ abgespeichert werden.

- Zusatzaufgabe: Erstellen sie Unterklassen für Dictionaries verschiedener Sprachen o. Themenbereiche