Examenopdracht Python – Corpuslinguïstiek

Hieronder vindt u de 6 deelopdrachten die samen de examenopdracht uitmaken voor het Python-onderdeel van het vak corpuslinguïstiek. Elke deelopdracht telt even sterk mee voor het eindtotaal in dit vakonderdeel. Deze opdrachten mogen opgelost worden met Python 2.x of Python 3.x maar wees consequent. Maak bij voorkeur gebruik van een open source editor naar keuze, zoals:

- *Sublime* (http://www.sublimetext.com/)
- *Idle* (http://docs.python.org/2/library/idle.html)
- *Text Wrangler* (http://www.barebones.com/products/textwrangler/)

De opdrachten moeten via email worden ingediend op **vrijdag 17 januari 2014**, ten laatste om middernacht, bij mike.kestemont@uantwerpen.be en steven.gillis@uantwerpen.be. Op **vrijdag 24 januari 2014, vanaf 14u** vindt de individuele, mondelinge bespreking (20 min.) van de eindopdracht plaats (Lange Winkelstraat 40-42, Kamer SL-306). Iedere opdracht hieronder moet resulteren in een afzonderlijke functie (volg secuur de naamgeving die hieronder wordt toegelicht). Plaats deze functies in één tekstbestand en dien dit in onder de naam <voornaam_familienaam.py>. Licht uw code waar nodig toe met trefzeker maar beknopt commentaar. U kan naar believen gebruik maken van online hulpmiddelen, maar het is niet toegelaten om met andere personen rechtstreeks over (de oplossing voor) deze opdracht te corresponderen. Als de lesgevers toch een dergelijke correspondentie kunnen achterhalen (bv. op online fora), zal dit onverbiddelijk resulteren in een nulresultaat voor het hele vak. Zorg er daarom voor dat u van elke lijn code perfect weet wat deze bijdraagt en kies steeds duidelijke namen voor uw variabelen.

Veel succes en alvast onze beste wensen voor 2014!

Deelopdracht 1: `load_words(fname, tolower)`

Deze functie moet de inhoud van een bestand met platte tekst kunnen inlezen en verwerken. Deze functie is van twee parameters afhankelijk: `fname` bevat het pad naar het tekstbestand; `tolower` is een *boolean* die aangeeft of de woorden in de woordenlijst al dan niet naar *lowercase* moeten worden omgezet. Alle tekens in `string.punctuation` moeten worden verwijderd uit de tekst via reguliere expressies. Hetzelfde geldt voor alle cijfers. Maak van reguliere expressies gebruik om vervolgens de tekst in woorden te splitsen aan weerszijden van (reeksen) witruimte. Verwijder zeker de witruimte aan het begin en einde van de tekst (gebruik hiervoor de geëigende functie!). Vermijd dat de woordenlijst "lege" woorden (*empty strings*) bevat. De functie moet een *list* teruggeven met de woorden die uit het tekstbestand geëxtraheerd werden. De functie kan dus als volgt worden aangesproken:

```
words = load_file("genesis.txt", True)
```

Deelopdracht 2: `rel_freq_dict(words)`

Deze functie krijgt als parameters een lijst met woorden (`words`). Vul eerst een Python *dictionary* die voor elk woord de absolute frequentie bevat. Bereken vervolgens het totaal aantal woorden in de *dictionary* en creëer vervolgens een nieuwe *dictionary* die de relatieve frequentie van elk woord bevat. De functie zal uiteindelijk de *dictionary* met deze relatieve frequenties teruggeven:

```
rel_freq = rel_freq_words(words)
```

Deelopdracht 3: `print_most_frequent(freq_dict, topn, reverse)`

Deze functie krijgt als parameter een *frequency dictionary* en zal de woorden hierin rangschikken op basis van hun relatieve frequentie. Laat de functie checken via `type()` of `freq_dict` wel een *dictionary* is en zorg dat er anders een `TypeError` wordt opgeworpen via `raise`. Op basis van de `Integer`-waarde die via de parameter `topn` wordt gespecifieerd, print de functie de `topn` meest frequente woorden op een aantrekkelijke wijze naar het scherm. Let er op dat je de woorden in de juiste volgorde rangschikt: de gebruiker van de functie moet via de booleaanse parameter `reverse` kunnen aangeven of er al dan niet van hoog naar laag moet gerangschikt worden: `reverse = True` betekent van hoog naar laag rangschikken; `reverse = False` betekent van laag naar hoog rangschikken. Deze functie heeft geen `return`-waarde:

```
print_most_frequent(rel_freq, 100, True)
```

Deelopdracht 4: `print_stats(words, output_file)`

Deze functie krijgt als parameter de woordenlijst (`words`) die we in de eerste deelopdracht hebben gegenereerd en het pad naar een (mogelijk nog onbestaand) bestand waar we het resultaat van onze analyses naartoe willen schrijven. De volgende statistieken moeten in de functie berekend worden en naar het output-bestand worden geschreven:

- Het aantal unieke woorden (via een `set`);
- De gemiddelde woordlengte, uitgedrukt in aantal letters per woord;
- Het langste woord, uitgedrukt in aantal letters per woord;
- Een lijst van woorden die exact één keer voorkomen;
- Het gemiddelde aantal klinkers (a, e, u, i, o) in een woord.
- De ratio (unieke woorden) / (alle woorden)

Zorg dat deze functie (die overigens geen `return`-waarde heeft) deze informatie op een aantrekkelijke manier naar het als parameter gespecifieerde bestand `output_file` schrijft:

```
print_stats(words, "my_output.txt")
```

Deelopdracht 5: `extract_ngrams(words, n)`

Zogenaamde ngrammen worden vaak gebruikt in de computerlinguïstiek: het gaat om niet-overlappende groepjes met een lengte *n* die uit een langere lijst van items kunnen worden geëxtraheerd. Dat klinkt ingewikkeld maar beschouw onderstaande voorbeelden van woord-ngrammen voor volgende voorbeeldzin: “in principio erat verbum et verbum erat apud deum”.

unigrammen (n=1)

“in”, “principio”, “erat”, “verbum”, “et”, “verbum”, “erat”, “apud”, “deum”

bigrammen (n=2)

“in principio”, “principio erat”, “’erat verbum”, “verbum et”, “et verbum”, “verbum erat”, “erat apud”, “apud deum”

trigrammen (n=3)

“in principio erat”, “principio erat verbum”, “erat verbum et”, “verbum et verbum”, “et verbum erat”, “verbum erat apud”, “erat apud deum”

enz…

Schrijf een functie die gegeven een woordenlijst `words` voor een willekeurige `n`, de ngrammen teruggeeft. Voor `n=3` zal de functie dus alle trigrammen van de woorden teruggeven. Je zal moeten itereren over de woordenlijst die aan de functie wordt meegegeven. Het zal daarbij vooral zaak zijn `IndexErrors` te vermijden en toch alle relevante ngrammen van begin tot eind te retourneren. Tot slot moet de functie de "gemiddelde herhalingsgraad" van de ngrammen berekenen: achterhaal hoe vaak een ngram gemiddeld voorkomt in deze tekst en print deze waarde op aantrekkelijke wijze naar het scherm. (Deze waarde hoeft niet te worden teruggegeven door de functie.)

```
ngrams = ngrams(words, 3) # extract trigrams
```

Deelopdracht 6: `word2cv(words)`

Schrijf een functie die een voor een willekeurige reeks woorden (i.e. `words`) een naïeve grafeem-foneemconversie uitvoert: de functie creëert voor elk woord een *dependent tier*, oftewel een *tuple* (geen *list*!) die voor elke medeklinker in het woord een 'C' bevat en voor elke klinker (e/E, a/A, i/I, u/U, o/O) een 'V'. Merk op dat elke ononderbroken reeks van (mede)klinkers slecht door één 'C' of 'V' mag worden voorgesteld:

Voorbeelden:

```
mat       >    ('C','V','C')
maat      >    ('C','V','C')
matt      >    ('C','V','C')
waeld     >    ('C','V','C')
```

Elk oorspronkelijk woord in `words` wordt samen met de corresponderende *dependent tier* op een leesbare manier weggeschreven naar een apart tekstbestand dat de naam heeft `<word>_tieranalysis.txt`. De functie zelf geeft een *tuple* terug van alle *tiers* die werden gevonden. Check voor het teruggeven of de lengte van deze *tuple* gelijk is aan die van de oorspronkelijke `words`.

```
word2cv(["matt", "maat", "mat", ..., "waeld"])
```
