Programming for Computational Linguistics 2025/2026

Exercise Set 6 (2025-11-27)
Dictionaries

Once you have completed all exercises, send an email to the progclgrader@ims.uni-stuttgart.de with the title "submit set6", and with your python files as attachments. You should receive a response with feedback and a grade.

Exercises are due on **2025-12-16 23:59**, but we encourage you to do them during the lab session.

Exercise 1. We will say that a word "can be composed" from another word if it is possible to select some of the letters from the second word (maybe all of them) to build the first one (possibly changing the order). For example, word "python" can be composed from words "pantyhose", "immunotherapy", or "hypnotists". It can not be composed from word "ponytail" because "ponytail" does not have a letter "h". Similarly, word "lesson" can be composed from "professional" or "responsible", but it can not be composed from word "longest" as "longest" does not have enough "s" letters.

Your task will be to write a function `can_be_composed(word1, word2)` to check if `word1` can be composed from `word2`. Put your function into a file `compose.py`. Example usages:

- `can_be_composed("python", "pantyhose")` returns `True`
- `can_be_composed("python", "ponytail")` returns `False`
- `can_be_composed("code", "docent")` returns `True`
- `can_be_composed("messy", "reassembly")` returns `True`
- `can_be_composed("computational", "alphanumeric")` returns `False`
- `can_be_composed("linguistics", "criminologists")` returns `False`

Hint: Write a function `word2dict(word)` which will convert `word` to a dictionary with an information how many times every letter occured in `word`. Use `word2dict` in your `can_be_composed` function.

In the exercises below your task will be to write a part-of-speech tagger. The tagger should assign the most frequent POS tags to words. It should:

- read the training data in `CoNLL-U` format `http://universaldependencies.org/format.html`
- remember what is the most common POS tag for every word (capitalization should not matter, treat words "this" and "This" as the same word)
- tag the test words with the most frequent POS tags (in case of words for which more than one POS tag is the most frequent, select any of them)

For example, after reading `small_train.conllu` as training data (see on ILIAS or on the next page) your program should:

- tag word "the" as DET (it appears only with this tag)
- tag word "this" as DET (it appears twice as DET and only once as PRON)
- tag word "wish" as NOUN or VERB (it appears once as VERB and once as NOUN and you can select any of the options)
- tag word "my" as UNK (unknown, because it does not appear in the training data)

Exercise 2. Create a module `freq_tagger.py`. Within the module create an auxiliary function `most_common()`. The function should take as it's argument a dictionary with frequencies and return the tag with the highest frequency:

- `most_common({"NOUN": 2, "VERB": 1})` returns `"NOUN"`
- `most_common({"NOUN": 2 })` returns `"NOUN"`
- `most_common({"NOUN": 2, "DET": 5, "ADP": 1 })` returns `"DET"`

Exercise 3. Within the module `freq_tagger.py` create an auxiliary function `read_conllu()`. The function should take as an argument path to a file in CoNLL-U format and return a list of tuples of words and their POS tags. For example, `read_conllu("small_train.conllu")` should return a list of 37 tuples starting:
`[ ("The", "DET"), ("hottest", "ADJ"), ("item", "NOUN") ... ]`

Exercise 4. Within the module `freq_tagger.py` create a function `train_and_tag(train_file, test_words)`. The function should accept two arguments: `train_file` with a file with correct POS tags to train on (file in `CoNLL-U` format) and `test_words` to tag. Your function should return a list of POS tags for `test_words`. Example usages:

- ```
  train_and_tag("small_train.conllu",
                [ "This", "is", "my", "wish" ])
  ```

  returns `["DET", "AUX", "UNK", "NOUN"]`

- ```
  train_and_tag("small_train.conllu",
                [ "The", "citizens", "hate", "weapons",
                  "but", "love", "Christmas" ])
  ```

  returns `['DET', 'NOUN', 'UNK', 'NOUN', 'UNK', 'UNK', 'PROPN']`

- ```
  train_and_tag("en_gold.conllu",
                [ "The", "citizens", "hate", "weapons",
                  "but", "love", "Christmas" ])
  ```

  returns `["DET", "NOUN", "UNK", "NOUN", "CCONJ", "VERB", "PROPN"]`.

You can use all the auxiliary functions but you do not have to.
  ```

File `small_train.conllu`:

```
1   The        _   DET   _ _ _ _ _ _
2   hottest    _   ADJ   _ _ _ _ _ _
3   item       _   NOUN  _ _ _ _ _ _
4   on         _   ADP   _ _ _ _ _ _
5   Christmas  _   PROPN _ _ _ _ _ _
6   wish       _   NOUN  _ _ _ _ _ _
7   lists      _   NOUN  _ _ _ _ _ _
8   this       _   DET   _ _ _ _ _ _
9   year       _   NOUN  _ _ _ _ _ _
10  is         _   AUX   _ _ _ _ _ _
11  nuclear    _   ADJ   _ _ _ _ _ _
12  weapons    _   NOUN  _ _ _ _ _ _
13  .          _   PUNCT _ _ _ _ _ _

1   This       _   PRON  _ _ _ _ _ _
2   is         _   AUX   _ _ _ _ _ _
3   unlike     _   ADP   _ _ _ _ _ _
4   the        _   DET   _ _ _ _ _ _
5   situation  _   NOUN  _ _ _ _ _ _
6   last       _   ADJ   _ _ _ _ _ _
7   year       _   NOUN  _ _ _ _ _ _
8   when       _   ADV   _ _ _ _ _ _
9   we         _   PRON  _ _ _ _ _ _
10  evacuated  _   VERB  _ _ _ _ _ _
11  citizens   _   NOUN  _ _ _ _ _ _

1   From       _   ADP   _ _ _ _ _ _
2   the        _   DET   _ _ _ _ _ _
3   AP         _   PROPN _ _ _ _ _ _
4   comes      _   VERB  _ _ _ _ _ _
5   this       _   DET   _ _ _ _ _ _
6   story      _   NOUN  _ _ _ _ _ _

1   I          _   PRON  _ _ _ _ _ _
2   wish       _   VERB  _ _ _ _ _ _
3   you        _   PRON  _ _ _ _ _ _
4   all        _   DET   _ _ _ _ _ _
5   of         _   ADP   _ _ _ _ _ _
6   the        _   DET   _ _ _ _ _ _
7   best       _   ADJ   _ _ _ _ _ _
```