Programming for Computational Linguistics 2025/2026

Exercise Set 7 (2025-12-04)
Object Oriented Programming

Once you have completed all exercises, send an email to the progclgrader@ims.uni-stuttgart.de with the title "submit set7", and with your python files as attachments. You should receive a response with feedback and a grade.

Exercises are due on **2026-01-13 23:59**, but we encourage you to do them during the lab session.

Exercise 1. In the file `conlltoken.py` create a class `ConLLToken`. The constructor of the class should accept four parameters: `form, lemma, pos, morph` and store them in attributes with the same names.
With an appropriate special function ensure that running `str(tok)` will print such format: `form,lemma,pos,morph`

```
tok1 = ConLLToken("comes", "come", "VERB",
                  "Mood=Ind|Number=Sing|Person=3")

tok2 = ConLLToken("years", "year", "NOUN",
                  "Number=Plur")

print("Token 1:", str(tok1))
print("Token 2:", tok2)
```

```
Token 1: comes,come,VERB,Mood=Ind|Number=Sing|Person=3
Token 2: years,year,NOUN,Number=Plur
```

Exercise 2. This exercise is a continuation of the previous one. Add to the class `ConLLToken` such methods:

- `is_punctuation` which returns `True` if the attribute `pos` is equal to `PUNCT` and `False` otherwise
- `is_inflected` which returns `False` if `lemma` is the same as `form` (capitalization should not count) and `True` otherwise
- `get_person` which returns the value for the field `Person` from `morph` (or `None` if it does not exist)

Example usage of your class:

```
tok1 = ConLLToken("comes", "come", "VERB",
                  "Mood=Ind|Number=Sing|Person=3")

tok2 = ConLLToken("year", "year", "NOUN",
                  "Number=Sing")


tok3 = ConLLToken(",", ",", "PUNCT", "_")

print("Punctuation?",
        tok1.is_punctuation(),
        tok2.is_punctuation(),
        tok3.is_punctuation())

print("Inflected?",
        tok1.is_inflected(),
        tok2.is_inflected(),
```

```
        tok3.is_inflected())

print("Person?",
        tok1.get_person(),
        tok2.get_person(),
        tok3.get_person())
```

```
Punctuation? False False True
Inflected? True False False
Person? 3 None None
```

Exercise 3. This exercise is a continuation of the previous one. In the file `builder.py` create two different classes responsible for building instances of `ConLLToken` tokens:

- class `ConLLUTokenBuilder` should have a function `buildToken(line)`. The function should accept a line in `CoNLL-U` format and return an instance of `ConLLToken` with appropriate values.
- class `ConLL09TokenBuilder` should have a function `buildToken(line)`. The function should accept a line in `CoNLL-09` format and return an instance of `ConLLToken` with appropriate values. Read about the `CoNLL-09` format here `https://ufal.mff.cuni.cz/conll2009-st/task-description.html` (we want fields `LEMMA`, `POS`, `FEAT`, and not `PLEMMA`, `PPOS`, `PFEAT`).

Example usage of your code (pay attention to the type of the returned object):

```
# checking ConLLUTokenBuilder
builder = ConLLUTokenBuilder()
line = "1 The the DET _ Definite=Def|PronType=Art _ _ _ _"
tok = builder.buildToken(line)

print("Token:", str(tok))
print("Type:", type(tok))
```

```
Token: The,the,DET,Definite=Def|PronType=Art
Type: <class 'conlltoken.ConLLToken'>
```

Download from ILIAS files `pos.py` and `small_train.conllu` (they should be in the same directory as your file `builder.py`). Analyze what is going on in the code. How the different builders are being created? How they are used? Finish the function `get_unique_pos` which should return a list of unique POS tags. Submit both `builder.py` and `pos.py`.

After fixing and running `pos.py` you should see:

```
Nr of unique POS (should be 10): 10
```