Programming for Computational Linguistics 2025/2026

Exercise Set 8 (2025-12-11)
Recursion

Once you have completed all exercises, send an email to the progclgrader@ims.uni-stuttgart.de with the title “submit set8”, and with your python files as attachments. You should receive a response with feedback and a grade.

Exercises are due on **2026-01-20 23:59**, but we encourage you to do them during the lab session.

Exercise 1. In a file `noloops.py`, define the following functions. Do not use any `for`, `while`, or list comprehensions in this file. Use recursion instead. Feel free to define as many extra functions as you need. An example for each case is given below.

- `product(nums)` accepts a list of numbers `nums` as a parameter, and returns the product of those numbers. The product of an empty list is one.

```
>>> product([2, 3, 1, -5, 3])
-90
>>> product([ ])
1
```

- `squares(nums)` accepts a list of numbers `nums` as a parameter, and returns a new list containing those numbers’ squares

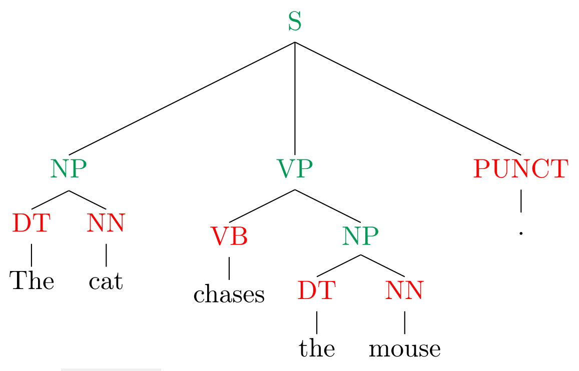
```
>>> squares([-2, -1, 0, 1, 2, 5])
[4, 1, 0, 1, 4, 25]
```

- `num_zeros(n)` accepts a positive(!) integer `n` as a parameter, and returns the number of zeros in that number’s decimal representation

```
>>> num_zeros(5)
0
>>> num_zeros(20)
1
>>> num_zeros(10101)
2
>> num_zeros(100200304501)
6
```

The file `constituents.py` contains the classes `Phrase` and `Token`. These classes can be used together to represent a constituency tree. You will use these classes for the following exercises.

Exercise 2. In a file called `catandmouse.py`, represent the following constituency tree using instances of the `Phrase` and `Token` classes. For clarity, phrase types are rendered in green, parts of speech are rendered in red, and token text is rendered in black.



The variable `sent` should contain a `Phrase` object corresponding to the topmost S phrase of the tree.

Exercise 3. In text corpora such as the Penn treebank, constituency trees are often represented as s-expressions. An s-expression is a text format for encoding trees and subtrees, using parenthesized lists. An s-expression representing the above tree would look like this:

```
(S
   (NP
      (DT The)
      (NN cat)
   )
   (VP
      (VB chases)
      (NP
         (DT the)
         (NN mouse)
      )
   )
   (PUNCT .)
)
```

Whitespace and newlines are for clarity only — the following s-expression is equivalent:

```
(S (NP (DT The) (NN cat)) (VP (VB chases) (NP (DT the) (NN mouse))) (PUNCT .))
```

Note that in s-expression form, the each parenthetical list corresponds to either a phrase or token. The first element of a list is either the phrase-type or part-of speech, and subsequent elements of a list are either a phrase's children, or a token's text.

In the file `constituents.py`, modify the classes `Phrase` and `Token`. To each class, add a `__str__` method which returns an s-expression corresponding to that phrase or token. For example:

```
>>> t_the = Token('DT', 'The')
>>> t_cat = Token('NN', 'cat')
>>> p_the_cat = Phrase('NP', [t_the, t_cat])
>>> str(t_the)
'(DT The)'
>>> str(t_cat)
'(NN cat)'
>>> str(p_the_cat)
'(NP (DT The) (NN cat))'
```

Whitespace is not important, so long as you produce a valid s-expression for each phrase and token.

Exercise 4. S-expressions can be represented in python using lists of (lists of (lists of (...))) strings. For example, the above s-expression would be represented in the following way in python:

```
['S',
  ['NP',
    ['DT', 'The'],
    ['NN', 'cat']
  ],
  ['VP',
    ['VB', 'chases'],
    ['NP',
      ['DT', 'the'],
      ['NN', 'mouse']
    ]
  ],
  ['PUNCT', '.']
]
```

In a file `list2parsetree.py`, write a function `list2parsetree(l)`. This function should accept as its input a `list` l, in the format described above. It should return the root node of a constituency tree made of `Phrase` and `Token` objects.