Problem set 1b Computational Linguistics Fall 2023

This problem set involves working concretely with grammars, parsing, and the associated data structures in NLTK.

In [ ]:
```
import nltk
from nltk.tree import Tree
```

1. In NLTK, create and display graphically Xbar trees in the style of Lecture 1 and Lecture Note 1 for the following three sentences. Define the trees ta, tb, and t3 and display them inline in your notebook. In case of ambiguity, choose the semantically plausible tree shape.

a. the title on every book under the table will have faded

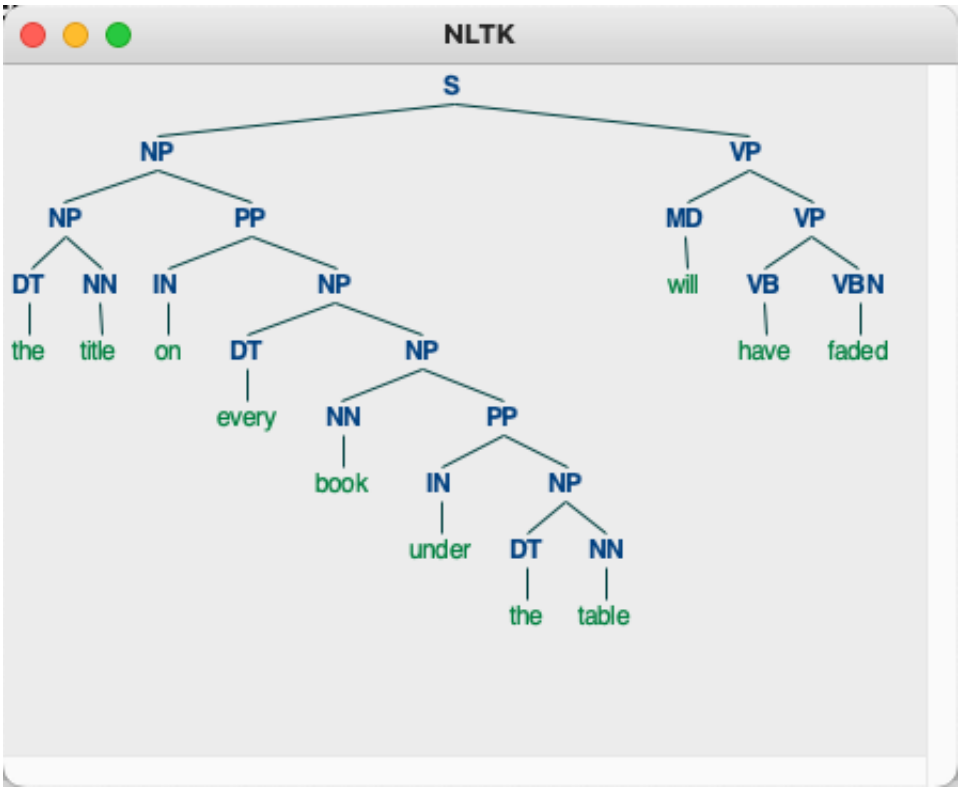
b. she wondered whether he knew that they believed they admired Keisha

c. every eligeble candidate told a presiding supervisor about it

In [ ]:
```
t1 = Tree.fromstring("\
    (S (NP \
        (NP (DT the) (NN title))\
        (PP (IN on) (NP\
            (DT every)\
            (NP (NN book) (PP\
                (IN under) (NP (DT the) (NN table))))))) \
       (VP (MD will) (VP (VB have) (VBN faded) )))\
    ")
```

In [ ]:
```
print(t1)
t1.draw()
```

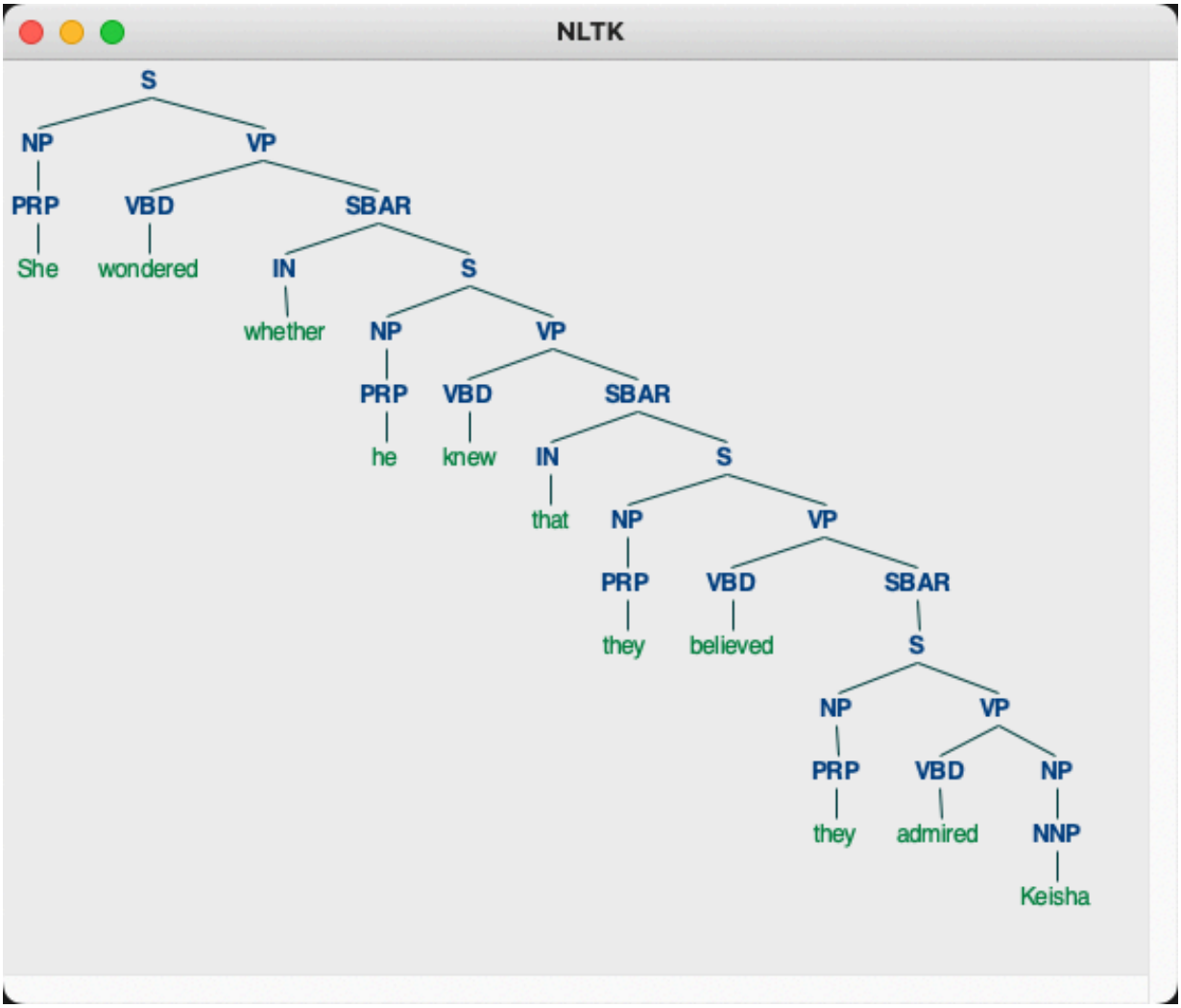
```
(S
  (NP
    (NP (DT the) (NN title))
    (PP
      (IN on)
      (NP
        (DT every)
        (NP (NN book) (PP (IN under) (NP (DT the) (NN table)))))))
  (VP (MD will) (VP (VB have) (VBN faded))))
```



In [ ]:
```
t2_s1 = Tree.fromstring("\
    (S (NP (PRP they))\
         (VP (VBD admired) (NP (NNP Keisha) )))")
t2_s2 = Tree.fromstring(f"\
    (S (NP (PRP they))\
         (VP (VBD believed) (SBAR {t2_s1} )))")
t2_s3 = Tree.fromstring(f"\
    (S (NP (PRP he))\
             (VP (VBD knew) (SBAR (IN that) {t2_s2} )))")
t2 = Tree.fromstring(f"\
    (S (NP (PRP She))\
             (VP (VBD wondered) (SBAR (IN whether) {t2_s3} )))")
```

In [ ]:
```
print(t2)
t2.draw()
```

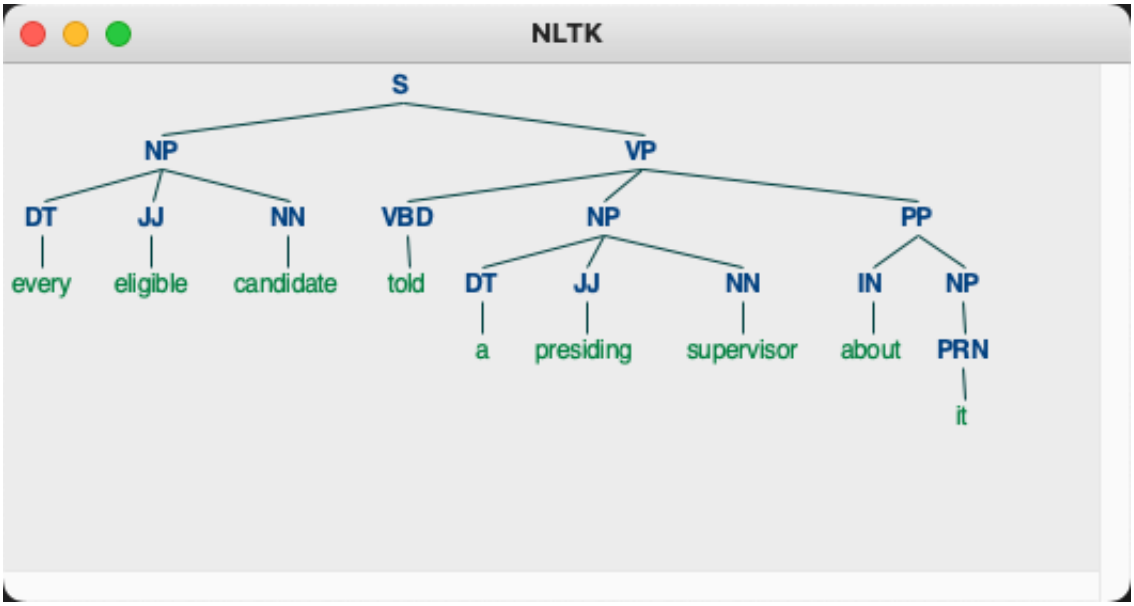
```
(S
  (NP (PRP She))
  (VP
    (VBD wondered)
    (SBAR
      (IN whether)
      (S
        (NP (PRP he))
        (VP
          (VBD knew)
          (SBAR
            (IN that)
            (S
              (NP (PRP they))
              (VP
                (VBD believed)
                (SBAR
                  (S
                    (NP (PRP they))
                    (VP (VBD admired) (NP (NNP Keisha))))))))))))
```



In [ ]:
```
np = Tree.fromstring("\
    (NP (DT every) (JJ eligible) (NN candidate))")
vp = Tree.fromstring("\
    (VP (VBD told) \
        (NP (DT a) (JJ presiding) (NN supervisor))\
        (PP (IN about) (NP (PRN it)) ))")
t3 = Tree.fromstring(f"\
    (S {np} {vp} )")
```

In [ ]:
```
print(t3)
t3.draw()
```

```
(S
  (NP (DT every) (JJ eligible) (NN candidate))
  (VP
    (VBD told)
    (NP (DT a) (JJ presiding) (NN supervisor))
    (PP (IN about) (NP (PRN it)))))
```



1. Grammar of transitive verbs and relative clauses.

The following are transitive sentences.

the electrician repaired the router

the plumber damaged the sink

the plumber admired the electrician

In the following, what are called subject relative clauses have been added to one of the NPs. This means that the relative pronoun "who" or "that" fills the subject position of the embedded verb.

the electrician who admired the plumber repaired the router

the plumber admired the electrician who damaged the sink

the plumber repaired the sink that admired the router

In the following, what are called object relative clauses have been added to one of the NPs. This means that the relative pronoun fills the object position of the embedded transitive verb.

the plumber who the electrician admires damaged the sink

the plumber damaged the sink that the electrician repaired.

More relative clauses can be added. In the second example, there are two relative on the noun "electrician".

the plumber who the electrician admires damaged the sink that the plumber repaired

the plumber admires the electrician who repaired the sink who the plumber admires

a. In NLTK, write a CFG in ***Chomsky normal form*** for sentences with this pattern. The terminal words should be the ones that are seen above. Name the grammar g2.

Start with the following grammar, and add productions. In the symbols, the character 'a' indicates animate, and the character 'i' indicates inanimate.

S -> NPa VP S -> NPi VP V -> 'admired' | 'envied' V -> 'damaged' | 'repaired' Na -> 'plumber' | 'electrician' Ni -> 'router' | 'sink' THE -> 'the' NPa -> THE Na NPi -> THE Ni VP -> V NPa VP -> V NPi WHO -> 'who' THAT -> 'that'

The grammar should impose the constraint that "electrician" and "plumber" go with the relative pronoun "who" and not the relative pronoun "that", while "sink" and "router" go with the relative pronoun "that" and not the relative pronoun "who". Use the distinction between 'a' and 'i' to impose this. As came up in class, this contraint is arguably not really valid, but assume it.

[NP the sink who the electrician damaged][NP the electrician that damaged the sink]

Assume that semantically odd sentences like the following are ok.

the sink admires the router the router repaired the plumber the plumber repaired the electrician

In [ ]:
```
x0 = """ S -> NPa VP
S -> NPi VP
V -> 'admired' | 'envied'
V -> 'damaged' | 'repaired'
Na -> 'plumber' | 'electrician'
Ni -> 'router' | 'sink'
THE -> 'the'
NPa -> THE Na
NPi -> THE Ni
VP -> V NPa
VP -> V NPi
WHO -> 'who'
THAT -> 'that'
"""
```

In [ ]:
```
x2 = x0 + """
NPa -> NPa WHO VP
NPi -> NPi THAT VP

V -> 'admires' | 'envies'
V -> 'damages' | 'repairs'

NPa -> NPa WHO NPa V
NPi -> NPi THAT NPa V
"""
g2 = nltk.CFG.fromstring(x2)
print(g2)
```

```
Grammar with 25 productions (start state = S)
    S -> NPa VP
    S -> NPi VP
    V -> 'admired'
    V -> 'envied'
    V -> 'damaged'
    V -> 'repaired'
    Na -> 'plumber'
    Na -> 'electrician'
    Ni -> 'router'
    Ni -> 'sink'
    THE -> 'the'
    NPa -> THE Na
    NPi -> THE Ni
    VP -> V NPa
    VP -> V NPi
    WHO -> 'who'
    THAT -> 'that'
    NPa -> NPa WHO VP
    NPi -> NPi THAT VP
    V -> 'admires'
    V -> 'envies'
    V -> 'damages'
    V -> 'repairs'
    NPa -> NPa WHO NPa V
    NPi -> NPi THAT NPa V
```

1. Consider this sentence. Don't worry about the semantic implausibility.

the the router that repaired the sink that the plumber admired damaged the router that the plumber repaired

Using your grammar g2, define the following in Python.

a list of strings sb that corresponds to the sentence above

a parser p2 based on grammar g2

a generator gb optained with p2 operating on sb

a list of trees lb corresponding to gb (if you exhaust the generator gb in defining lb, redefine gb). The elements should be NLTK trees.

the number nb of trees in lb

In [ ]:
```
parser = nltk.ChartParser(g2)
```

In [ ]:
```
sent = "the plumber admires the electrician who repaired the sink who the pl
s1 = sent.split()
gen1 = parser.parse(s1)
for t in gen1: print(t)
```

```
(S
  (NPa (THE the) (Na plumber))
  (VP
    (V admires)
    (NPa
      (NPa
        (NPa (THE the) (Na electrician))
        (WHO who)
        (VP (V repaired) (NPi (THE the) (Ni sink))))
      (WHO who)
      (NPa (THE the) (Na plumber))
      (V admires))))
```

1. sent9.txt (supplied) is our list of sentences of lenght 9 that are in the relative clause pattern.

Verify that your grammar g2 finds a tree for each of the sentences

In [ ]:
```
sents = open("sent9.txt", "r").read().splitlines()
try:
    for sent in sents:
        s1 = sent.split()
        gen1 = parser.parse(s1)
except:
    print(f"Error in sentence: {sent}")
else:
    print("All sentences parsed successfully")
```

```
All sentences parsed successfully
```

1. Call a grammar symbol x nullable in grammar g if there some tree licensed by g that has the terminal string [] (the empty list) and has the root label x.

In [ ]:
```
g1_list = [('S',['DP','VP']), # categorial rules
      ('DP',['D','NP']),
      ('DP',['NP']),
      ('DP',['Name']),
      ('DP',['Pronoun']),
      ('NP',['N']),
      ('NP',['N','PP']),
      ('VP',['V']),
      ('VP',['V','PP']),
      ('VP',['V','CP']),
      ('VP',['V','VP']),
      ('VP',['V','DP','PP']),
      ('VP',['V','DP','CP']),
      ('VP',['V','DP','VP']),
      ('VP',['V','DP']),
      ('VP',['AdvP','VP']),
      ('PP',['P']),
      ('PP',['P','DP']),
      ('AP',['A']),
      ('AP',['A','PP']),
      ('CP',['C','S']),
      ('AdvP',['Adv']),
      ('NP',['AP','NP']),
      ('AP',['AdvP','AP']),
      ('D',['the']), # now the lexical rules
      ('D',['a']),
      ('D',['some']),
      ('D',['every']),
      ('D',['one']),
      ('D',['two']),
      ('A',['gentle']),
      ('A',['clear']),
      ('A',['honest']),
      ('A',['compassionate']),
      ('A',['brave']),
```

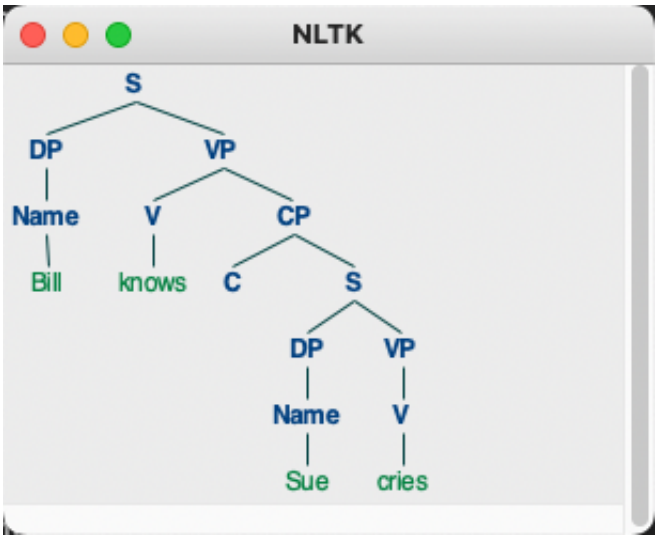
```
      ('A',['kind']),
      ('N',['student']),
      ('N',['teacher']),
      ('N',['city']),
      ('N',['university']),
      ('N',['beer']),
      ('N',['wine']),
      ('V',['laughs']),
      ('V',['cries']),
      ('V',['praises']),
      ('V',['criticizes']),
      ('V',['says']),
      ('V',['knows']),
      ('Adv',['happily']),
      ('Adv',['sadly']),
      ('Adv',['impartially']),
      ('Adv',['generously']),
      ('Name',['Bill']),
      ('Name',['Sue']),
      ('Name',['Jose']),
      ('Name',['Maria']),
      ('Name',['Presidents','Day']),
      ('Name',['Tuesday']),
      ('Pronoun',['he']),
      ('Pronoun',['she']),
      ('Pronoun',['it']),
      ('Pronoun',['him']),
      ('Pronoun',['her']),
      ('P',['in']),
      ('P',['on']),
      ('P',['with']),
      ('P',['by']),
      ('P',['to']),
      ('P',['from']),
      ('C',['that']),
      ('C',[]),
      ('C',['whether']),
      ('Coord',['and']),
      ('Coord',['or']),
      ('Coord',['but'])]

import td
```

a. In Stabler's grammar g1 (see g1.py, supplied), the symbol 'C' is nullable. Show the tree that verifies nullability.

In [ ]:
```
t5a_s = Tree.fromstring("\
    (S (DP (Name Sue)) (VP (V cries)))")
t5a = Tree.fromstring(f"\
    (S (DP (Name Bill)) (VP (V knows) (CP (C) {t5a_s} )))")
print(t5a)
t5a.draw()
```

```
(S
  (DP (Name Bill))
  (VP (V knows) (CP (C ) (S (DP (Name Sue)) (VP (V cries))))))
```



b. What are the nullable symbols in grammar gn (supplied)? Show the trees that verify nullability.

In [ ]:
```
xn = """
S -> A S
S -> B S
S -> C S
S -> C
A -> 'a'
A ->
B -> 'b'
C -> 'c'
B -> A A
"""
```

symbol A, B are nullable

In [ ]:
```
t5b_1 = Tree.fromstring("\
    (S (A ) (S (C c)))")
print(t5b_1)
t5b_1.draw()

t5b_2 = Tree.fromstring("\
    (S (B (A ) (A )) (S (C c)))")
print(t5b_2)
t5b_2.draw()
```

```
(S (A ) (S (C c)))
(S (B (A ) (A )) (S (C c)))
```

c. Describe in words a definition of 'nullable in g' along these lines:

```
symbol x is nullable in g iff there is a production x -> rhs
in g such that one of the following conditions are met
    1. rhs is [] (empty)
    2. every element in rhs is nullable
```

(mathematical induction)

d. Define in python a function 'nullable' which when appled to a grammar in NLTK format returns the set of nullable symbols.

Hint: start with the assumption that no symbols are nullable, then iteratively search for nullable symbols by looking at the productions. A skeleton that does something silly is supplied.

In [ ]:
```
gn = nltk.CFG.fromstring(xn)
gn.productions()
```

Out[ ]:
```
[S -> A S,
 S -> B S,
 S -> C S,
 S -> C,
 A -> 'a',
 A -> ,
 B -> 'b',
 C -> 'c',
 B -> A A]
```

In [ ]:
```python
def nullable(g):
    set = {p.lhs() for p in g.productions() if p.rhs() == ()}

    # repeat
    while(True):
        initial_len = len(set)
        for p in g.productions():
            flag = True
            # print(p)
            for r in p.rhs():
                # print(r)
                if(r not in set):
                    # print(f"not in list: {r}")
                    flag = False
                    break
            if(flag): set.add(p.lhs())

        # until no new elements are added to the set
        if len(set) == initial_len:
            break

    return set
```

e. Test your function on some sample grammars including gn.

In [ ]:
```python
nullable(gn)
```

Out[ ]:
```
{A, B}
```

In [ ]:
```python
x1 = """
S -> A S
S -> B S
S -> C S
S -> C
A -> 'a'
A ->
B -> 'b'
C -> 'c'
B -> A A
S -> D S
D -> B
"""
g5 = nltk.CFG.fromstring(x1)
g5.productions()
nullable(g5)
```

Out[ ]:
```
{A, B, D}
```