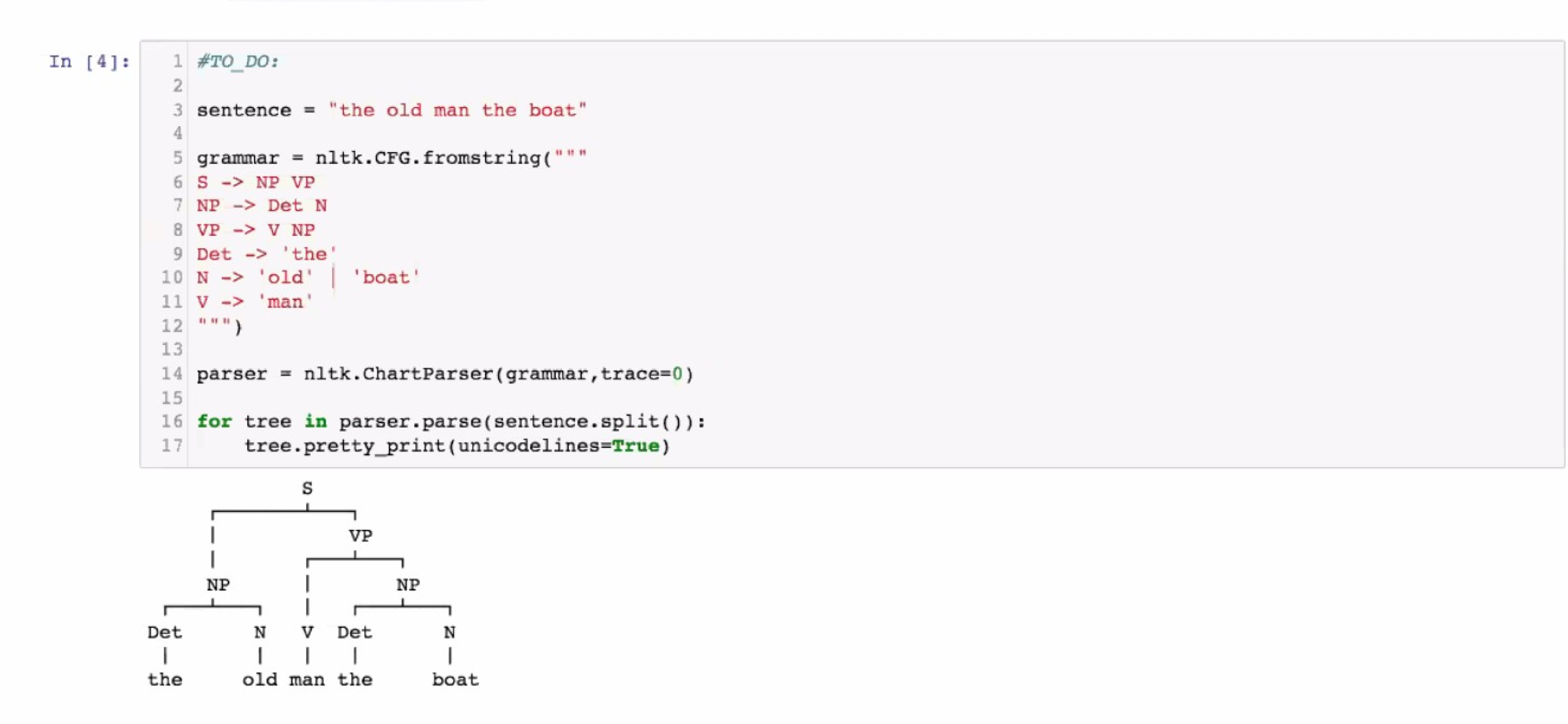
1

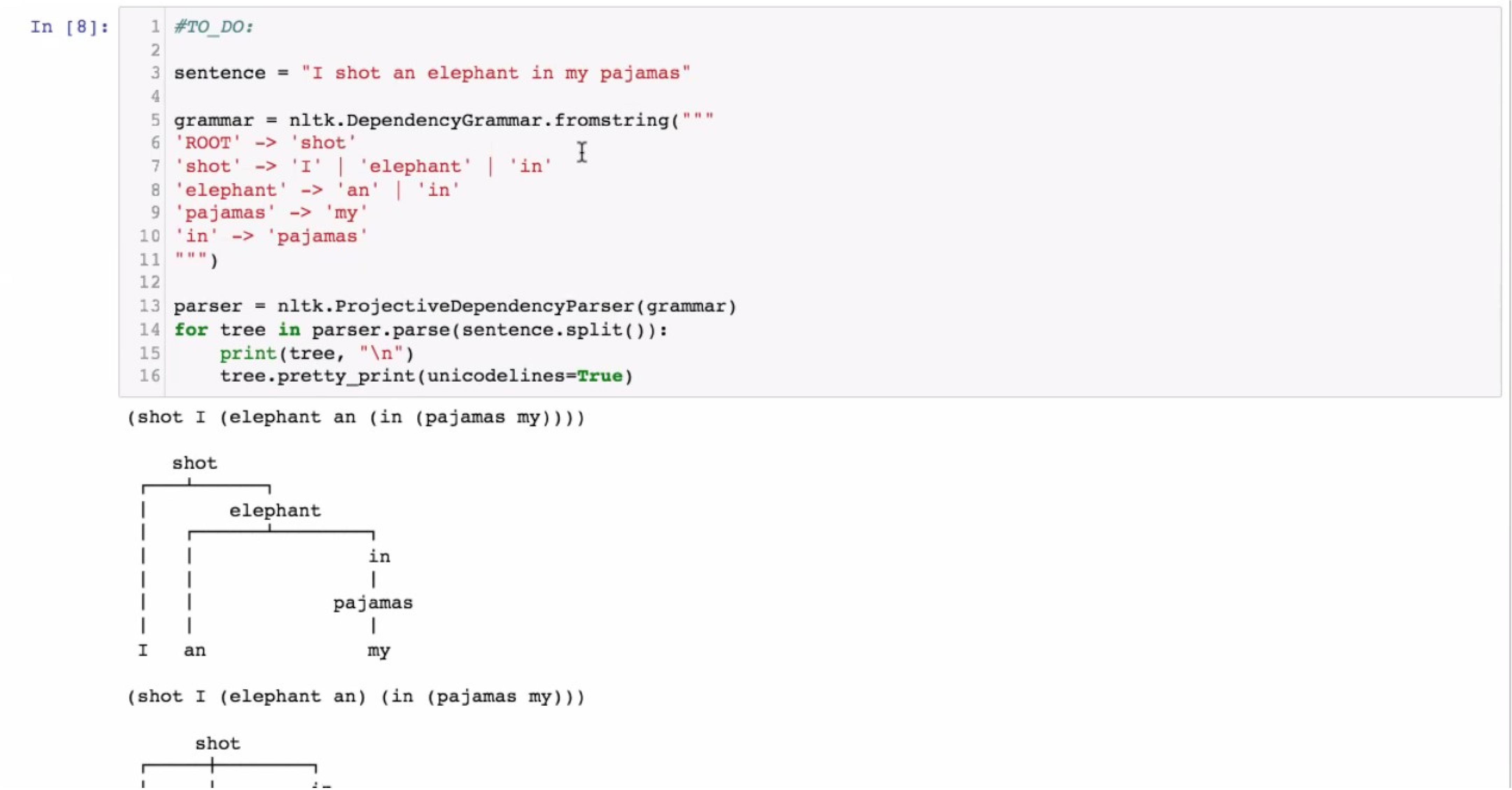


In [4]:

```
#TO_DO:

sentence = "the old man the boat"

grammar = nltk.CFG.fromstring("""
S -> NP VP
NP -> Det N
VP -> V NP
Det -> 'the'
N -> 'old' | 'boat'
V -> 'man'
""")

parser = nltk.ChartParser(grammar,trace=0)

for tree in parser.parse(sentence.split()):
    tree.pretty_print(unicodelines=True)
```

```
             S
     ┌───────┴───┐
     │           VP
     │       ┌───┴───┐
     NP      │       NP
 ┌───┴───┐   │   ┌───┴───┐
Det      N   V  Det      N
 │       │   │   │       │
the     old man the     boat
```

2

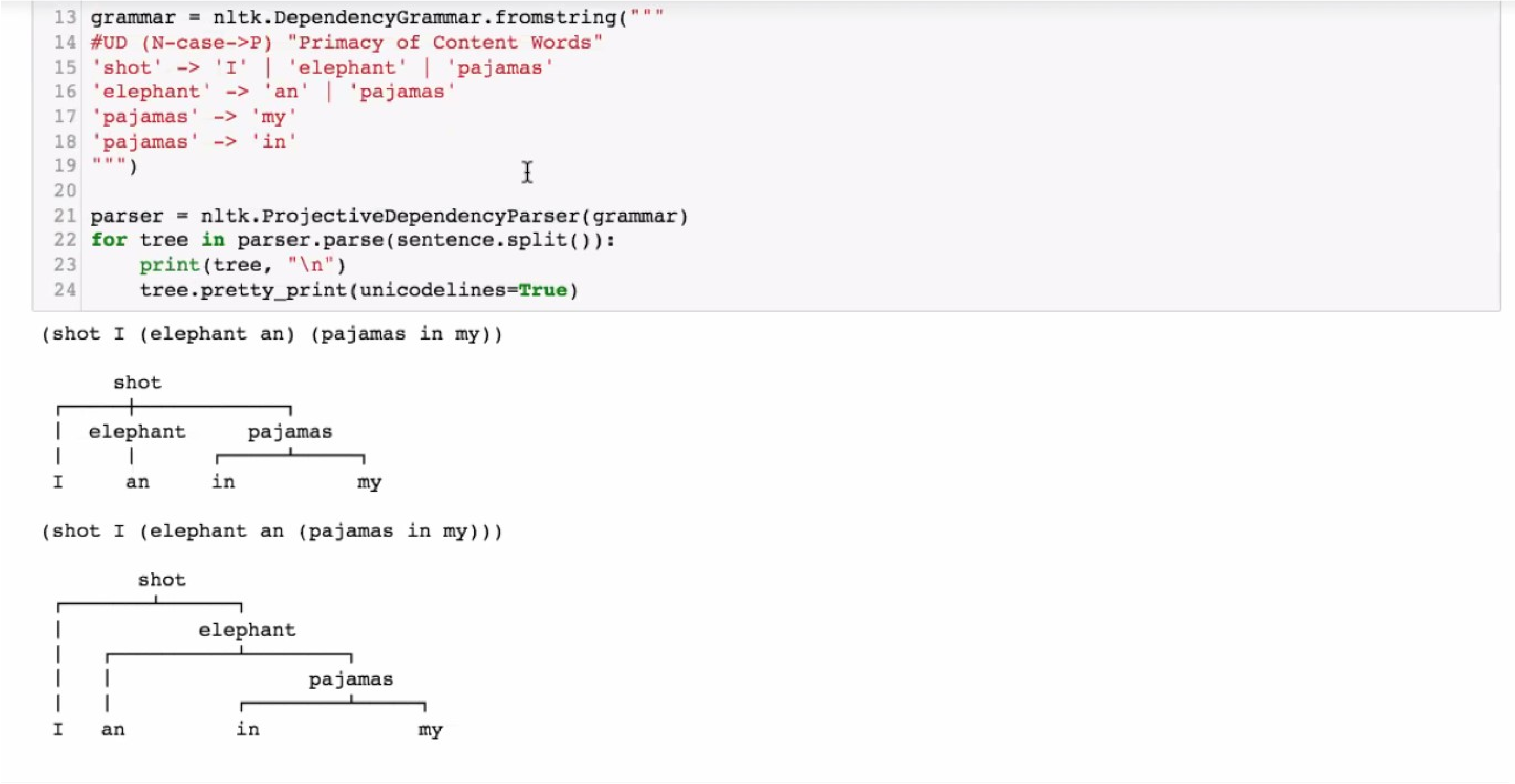


In [8]:

```python
#TO_DO:

sentence = "I shot an elephant in my pajamas"

grammar = nltk.DependencyGrammar.fromstring("""
'ROOT' -> 'shot'
'shot' -> 'I' | 'elephant' | 'in'
'elephant' -> 'an' | 'in'
'pajamas' -> 'my'
'in' -> 'pajamas'
""")

parser = nltk.ProjectiveDependencyParser(grammar)
for tree in parser.parse(sentence.split()):
    print(tree, "\n")
    tree.pretty_print(unicodelines=True)
```

```
(shot I (elephant an (in (pajamas my))))

    shot
 ┌───┴──────┐
 │       elephant
 │   ┌──────┴────────┐
 │   │               in
 │   │               │
 │   │            pajamas
 │   │               │
 I   an              my

(shot I (elephant an) (in (pajamas my)))

      shot
 ┌─────┼────────┐
```

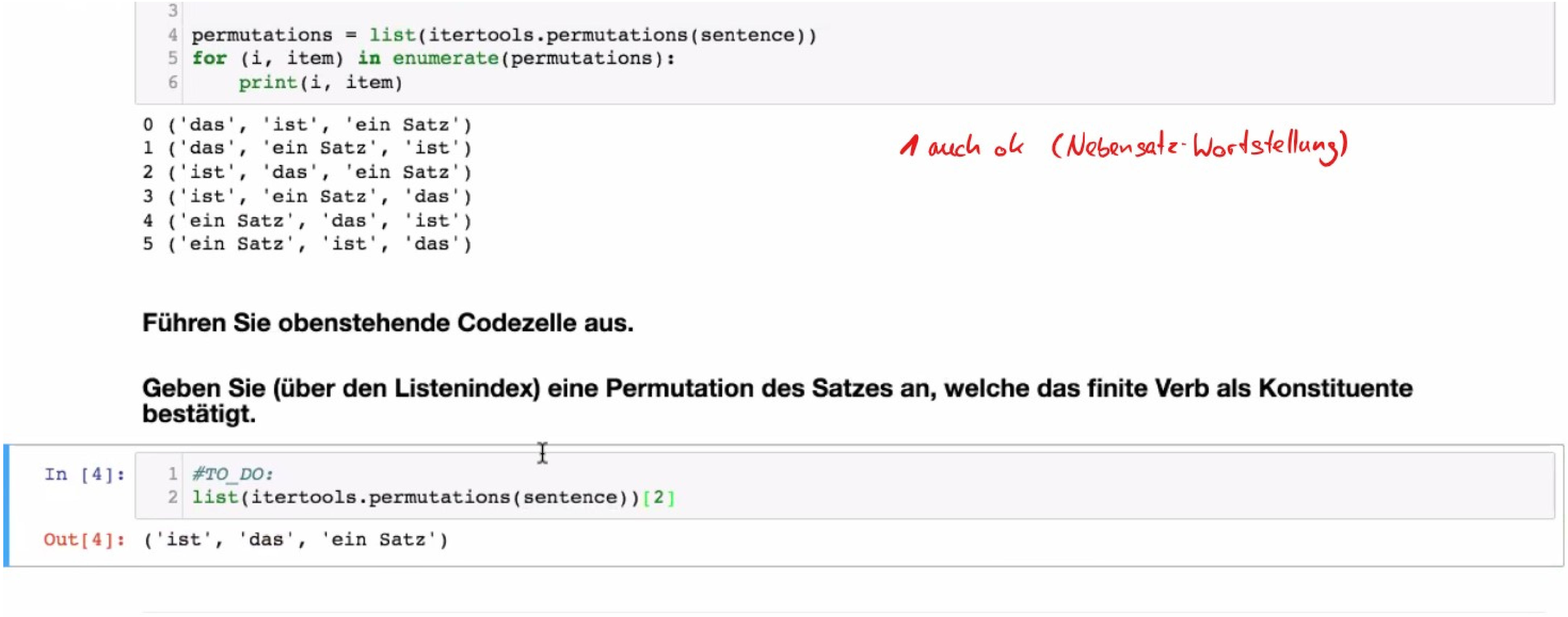
2 (UD-Scheme)



```python
grammar = nltk.DependencyGrammar.fromstring("""
#UD (N-case->P) "Primacy of Content Words"
'shot' -> 'I' | 'elephant' | 'pajamas'
'elephant' -> 'an' | 'pajamas'
'pajamas' -> 'my'
'pajamas' -> 'in'
""")

parser = nltk.ProjectiveDependencyParser(grammar)
for tree in parser.parse(sentence.split()):
    print(tree, "\n")
    tree.pretty_print(unicodelines=True)
```

```
(shot I (elephant an) (pajamas in my))

      shot
 ┌─────┼────────────┐
 │  elephant     pajamas
 │     │      ┌─────┴─────┐
 I     an     in          my

(shot I (elephant an (pajamas in my)))

        shot
 ┌───────┴──────┐
 │           elephant
 │   ┌──────────┴────────┐
 │   │                pajamas
 │   │          ┌────────┴─────┐
 I   an         in             my
```

3.1



```

permutations = list(itertools.permutations(sentence))
for (i, item) in enumerate(permutations):
    print(i, item)
```

```
0 ('das', 'ist', 'ein Satz')
1 ('das', 'ein Satz', 'ist')
2 ('ist', 'das', 'ein Satz')
3 ('ist', 'ein Satz', 'das')
4 ('ein Satz', 'das', 'ist')
5 ('ein Satz', 'ist', 'das')
```

1 auch ok (Nebensatz-Wortstellung)

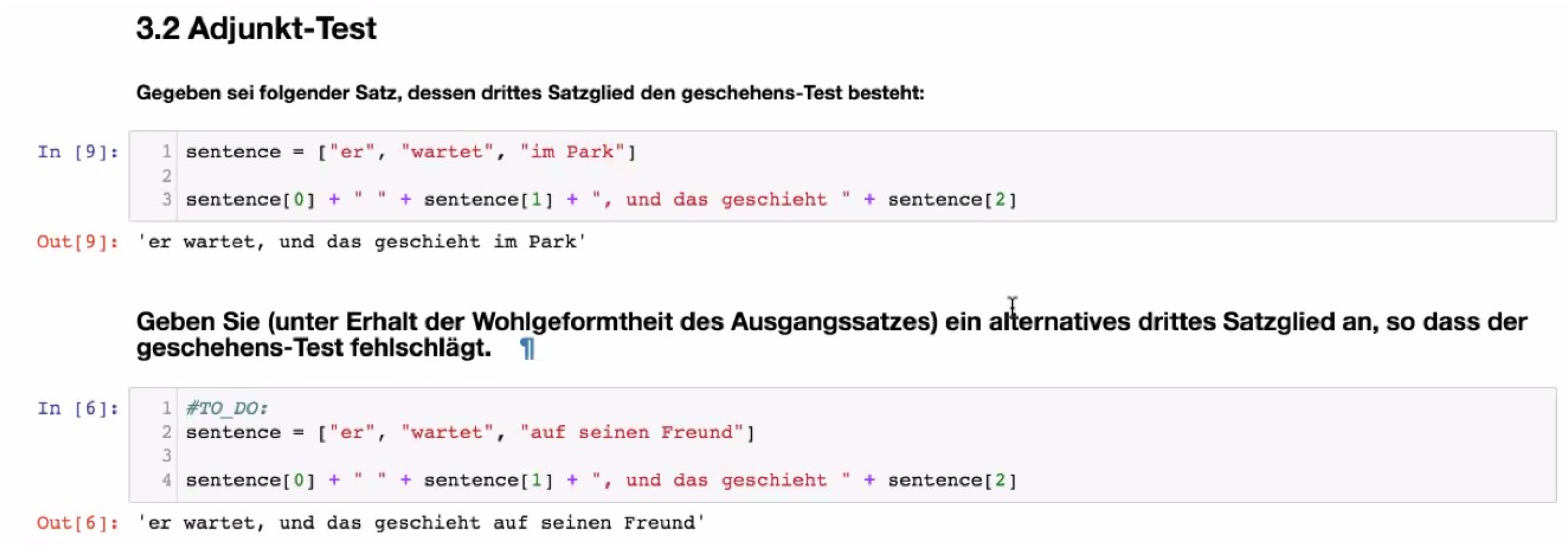
Führen Sie obenstehende Codezelle aus.

Geben Sie (über den Listenindex) eine Permutation des Satzes an, welche das finite Verb als Konstituente bestätigt.

```
In [4]:
#TO_DO:
list(itertools.permutations(sentence))[2]
```

```
Out[4]: ('ist', 'das', 'ein Satz')
```

3.2



3.2 Adjunkt-Test

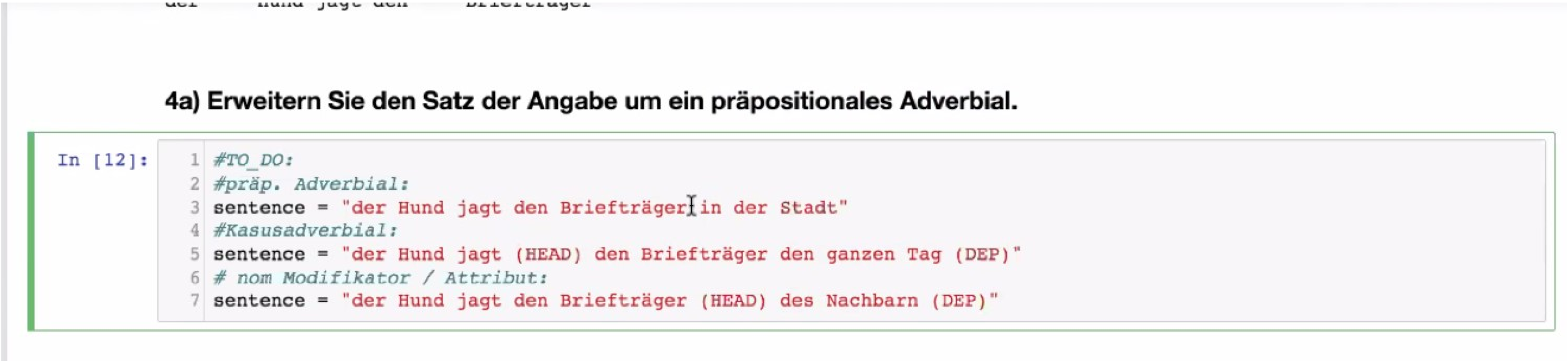
Gegeben sei folgender Satz, dessen drittes Satzglied den geschehens-Test besteht:

In [9]:
```
sentence = ["er", "wartet", "im Park"]

sentence[0] + " " + sentence[1] + ", und das geschieht " + sentence[2]
```
Out[9]:
```
'er wartet, und das geschieht im Park'
```

Geben Sie (unter Erhalt der Wohlgeformtheit des Ausgangssatzes) ein alternatives drittes Satzglied an, so dass der geschehens-Test fehlschlägt. ¶

In [6]:
```
#TO_DO:
sentence = ["er", "wartet", "auf seinen Freund"]

sentence[0] + " " + sentence[1] + ", und das geschieht " + sentence[2]
```
Out[6]:
```
'er wartet, und das geschieht auf seinen Freund'
```

4a) + Varianten



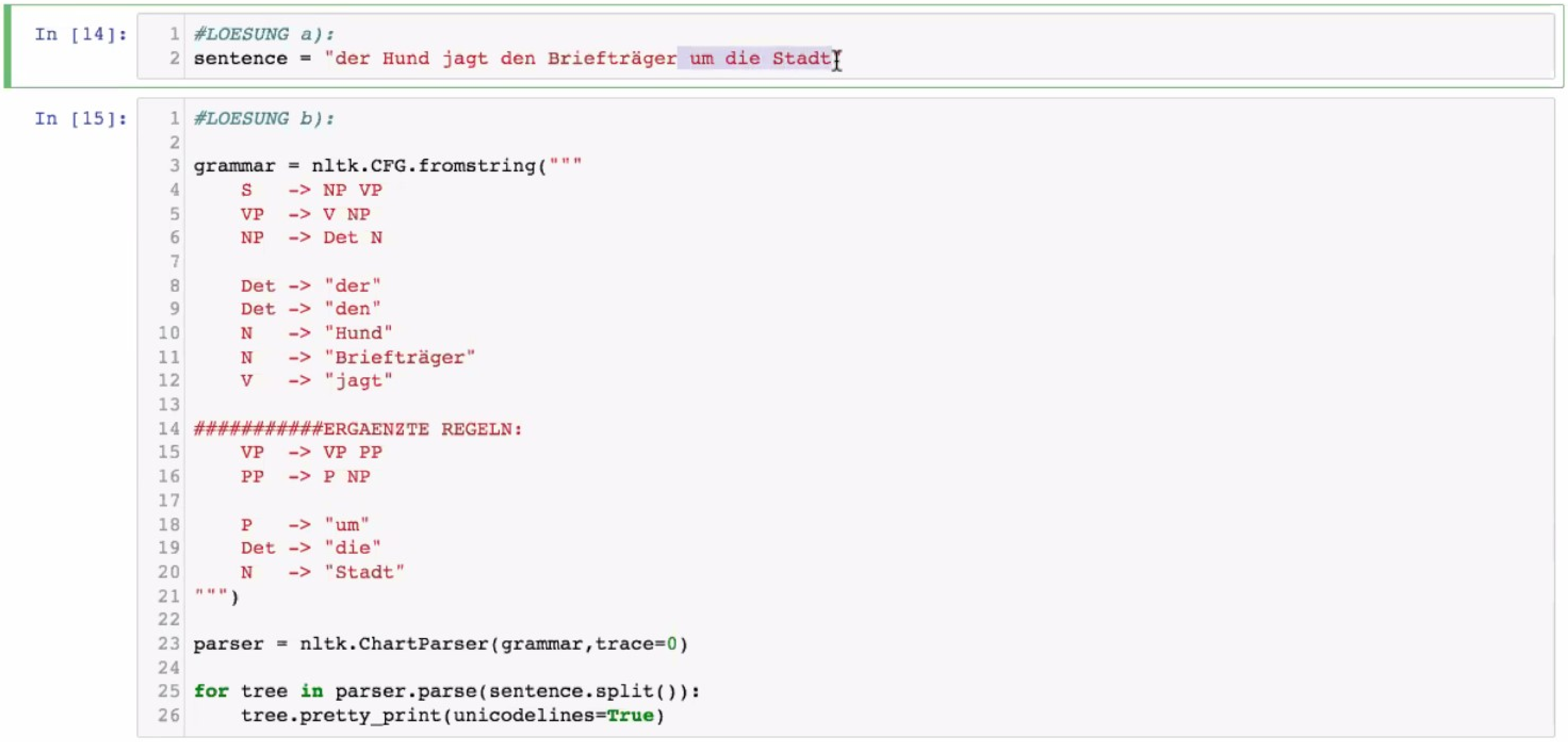
der Hund jagt den Briefträger

4a) Erweitern Sie den Satz der Angabe um ein präpositionales Adverbial.

In [12]:

```
#TO_DO:
#präp. Adverbial:
sentence = "der Hund jagt den Briefträger in der Stadt"
#Kasusadverbial:
sentence = "der Hund jagt (HEAD) den Briefträger den ganzen Tag (DEP)"
# nom Modifikator / Attribut:
sentence = "der Hund jagt den Briefträger (HEAD) des Nachbarn (DEP)"
```

4 a) b)



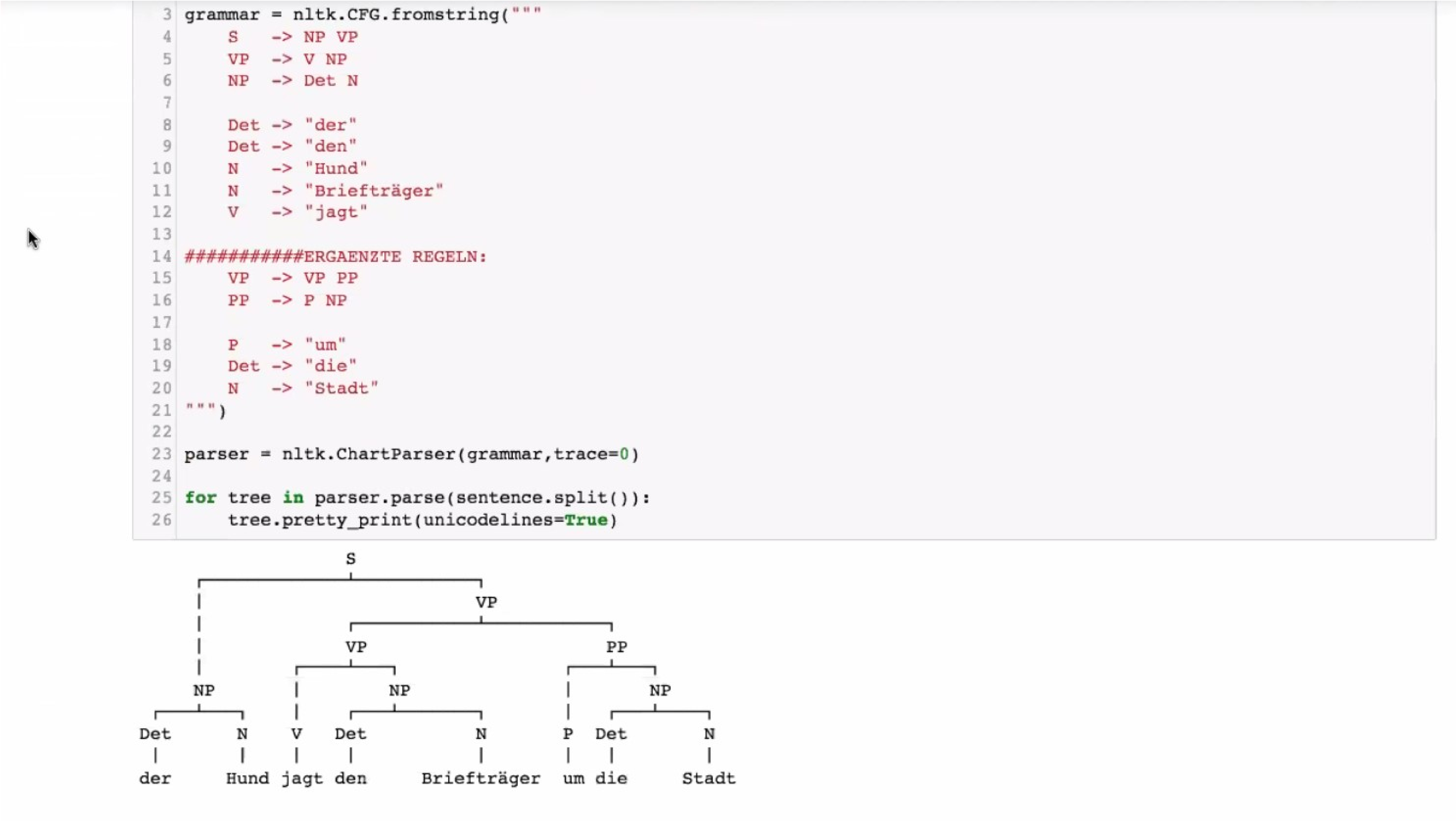
In [14]:

```
#LOESUNG a):
sentence = "der Hund jagt den Briefträger um die Stadt"
```

In [15]:

```
#LOESUNG b):

grammar = nltk.CFG.fromstring("""
    S   -> NP VP
    VP  -> V NP
    NP  -> Det N

    Det -> "der"
    Det -> "den"
    N   -> "Hund"
    N   -> "Briefträger"
    V   -> "jagt"

###########ERGAENZTE REGELN:
    VP  -> VP PP
    PP  -> P NP

    P   -> "um"
    Det -> "die"
    N   -> "Stadt"
""")

parser = nltk.ChartParser(grammar,trace=0)

for tree in parser.parse(sentence.split()):
    tree.pretty_print(unicodelines=True)
```

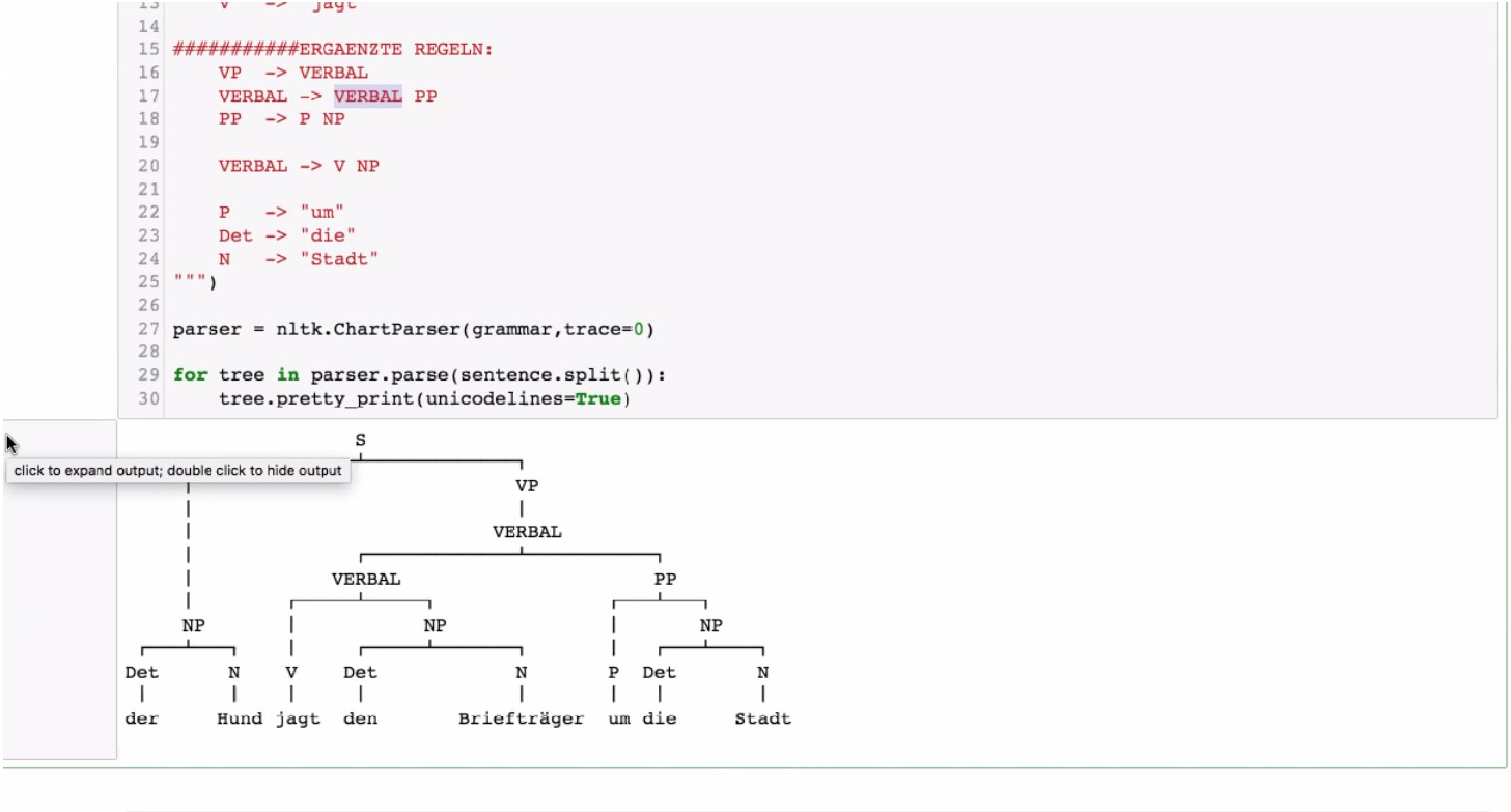
4



```python
grammar = nltk.CFG.fromstring("""
    S   -> NP VP
    VP  -> V NP
    NP  -> Det N

    Det -> "der"
    Det -> "den"
    N   -> "Hund"
    N   -> "Briefträger"
    V   -> "jagt"

###########ERGAENZTE REGELN:
    VP  -> VP PP
    PP  -> P NP

    P   -> "um"
    Det -> "die"
    N   -> "Stadt"
""")

parser = nltk.ChartParser(grammar,trace=0)

for tree in parser.parse(sentence.split()):
    tree.pretty_print(unicodelines=True)
```

```
                   S
     ┌─────────────┴───────────┐
     │                         VP
     │             ┌───────────┴───────────┐
     │             VP                      PP
     │        ┌────┴───┐               ┌───┴───┐
     NP       │        NP              │       NP
 ┌───┴───┐    │    ┌───┴───────┐       │   ┌───┴────┐
Det      N    V   Det          N       P  Det       N
 │       │    │    │           │       │   │        │
der     Hund jagt den     Briefträger  um die     Stadt
```

4 mit X-Bar

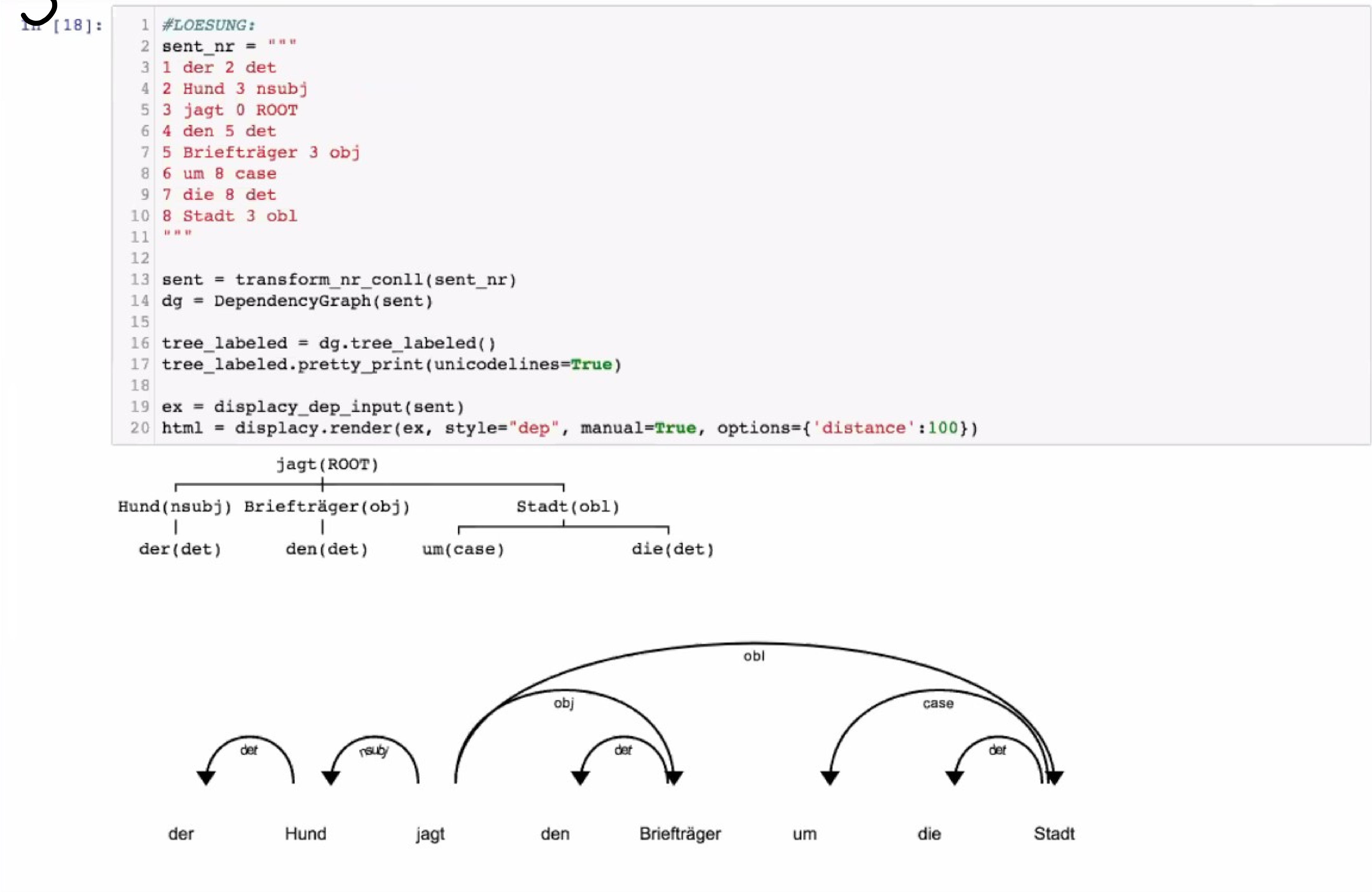


```
    V   -> jagt

##########ERGAENZTE REGELN:
    VP   -> VERBAL
    VERBAL -> VERBAL PP
    PP   -> P NP

    VERBAL -> V NP

    P    -> "um"
    Det  -> "die"
    N    -> "Stadt"
""")

parser = nltk.ChartParser(grammar,trace=0)

for tree in parser.parse(sentence.split()):
    tree.pretty_print(unicodelines=True)
```

click to expand output; double click to hide output

```
                    S
     ┌──────────────┴─────────────┐
     │                            VP
     │                            │
     │                          VERBAL
     │              ┌─────────────┴───────────┐
     │            VERBAL                      PP
     │        ┌─────┴─────┐               ┌───┴───┐
     NP       │           NP              │       NP
 ┌───┴───┐    │    ┌──────┴───────┐       │  ┌────┴────┐
Det      N    V    Det            N       P  Det       N
 │       │    │    │              │       │  │         │
der     Hund jagt  den       Briefträger  um die     Stadt
```

5

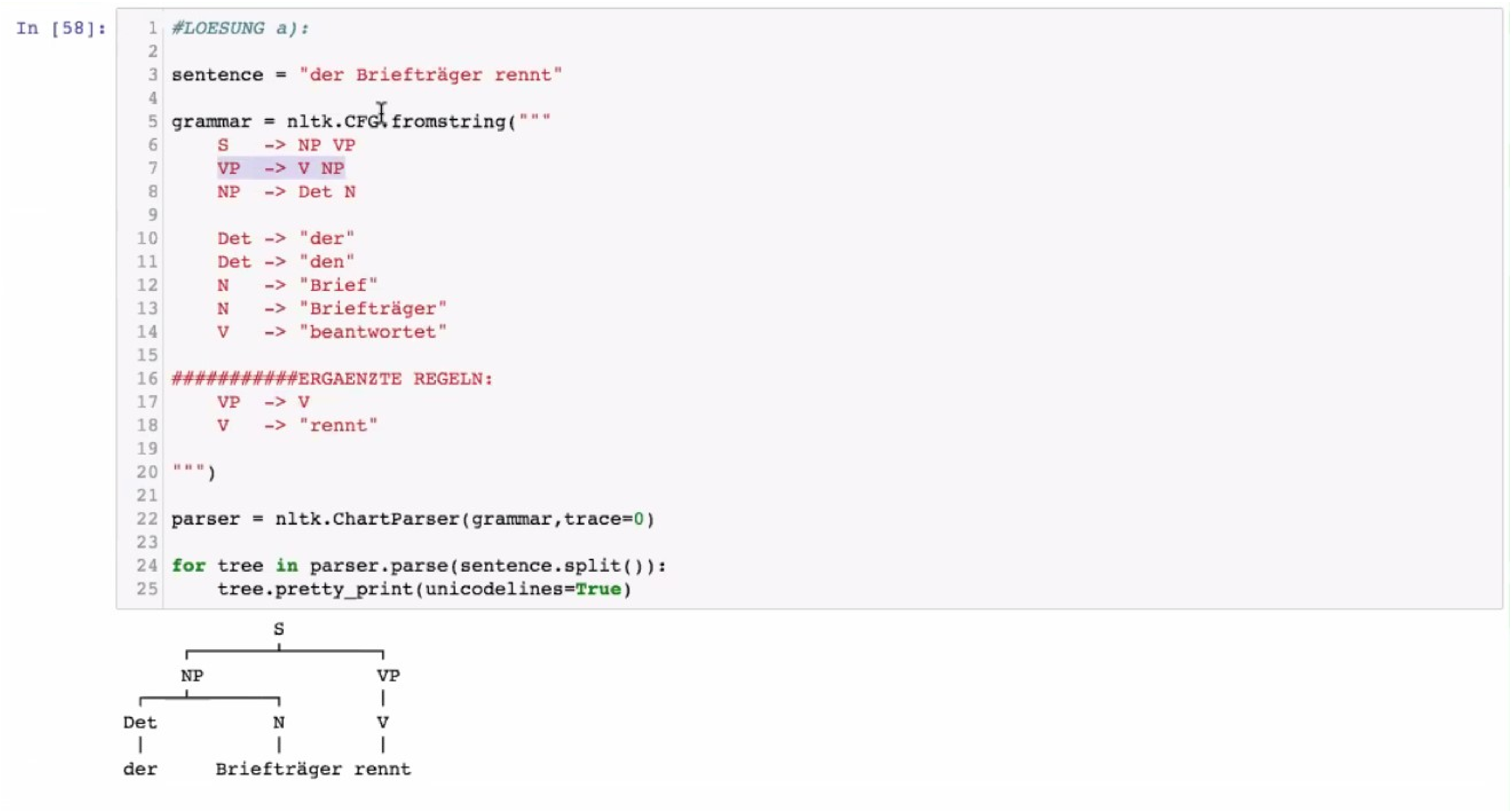


In [18]:

```
#LOESUNG:
sent_nr = """
1 der 2 det
2 Hund 3 nsubj
3 jagt 0 ROOT
4 den 5 det
5 Briefträger 3 obj
6 um 8 case
7 die 8 det
8 Stadt 3 obl
"""

sent = transform_nr_conll(sent_nr)
dg = DependencyGraph(sent)

tree_labeled = dg.tree_labeled()
tree_labeled.pretty_print(unicodelines=True)

ex = displacy_dep_input(sent)
html = displacy.render(ex, style="dep", manual=True, options={'distance':100})
```

```
                jagt(ROOT)
     ┌──────────────┼───────────────────────┐
Hund(nsubj) Briefträger(obj)            Stadt(obl)
     │              │             ┌─────────┴─────────┐
  der(det)       den(det)     um(case)             die(det)
```

6a)

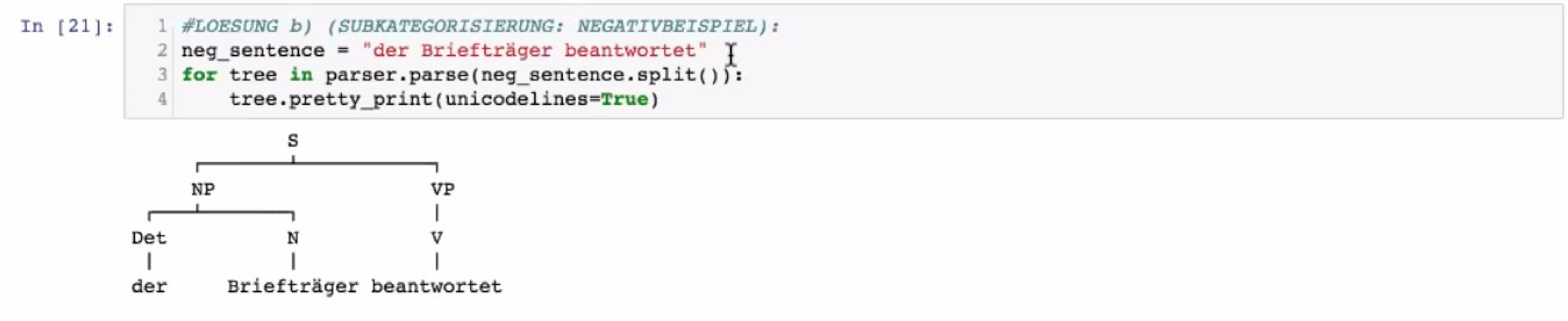


In [58]:

```
#LOESUNG a):

sentence = "der Briefträger rennt"

grammar = nltk.CFG.fromstring("""
    S   -> NP VP
    VP  -> V NP
    NP  -> Det N

    Det -> "der"
    Det -> "den"
    N   -> "Brief"
    N   -> "Briefträger"
    V   -> "beantwortet"

###########ERGAENZTE REGELN:
    VP  -> V
    V   -> "rennt"

""")

parser = nltk.ChartParser(grammar,trace=0)

for tree in parser.parse(sentence.split()):
    tree.pretty_print(unicodelines=True)
```

```
             S
     ┌───────┴────────┐
     NP               VP
┌────┴───────┐        │
Det          N        V
│            │        │
der     Briefträger rennt
```

6b)

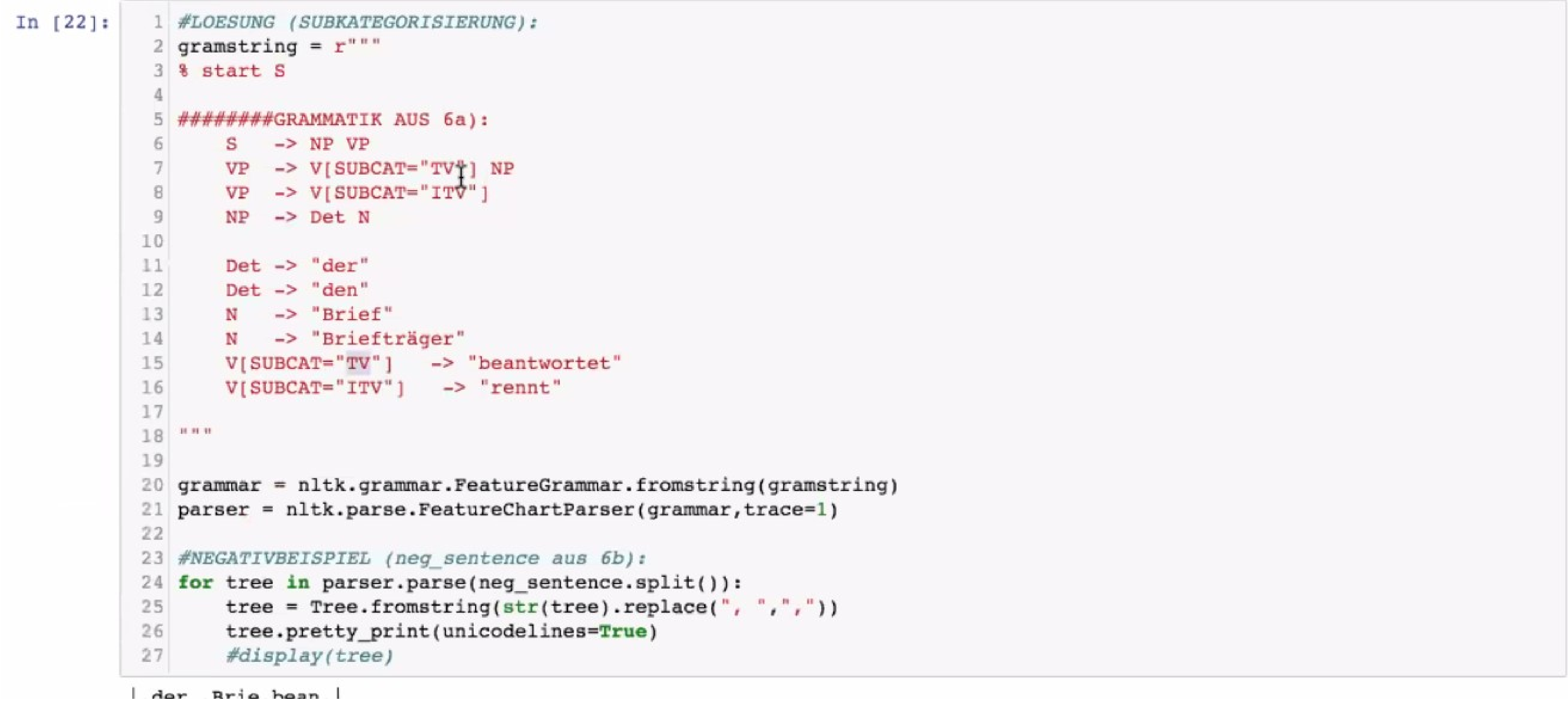


In [21]:

```
#LOESUNG b) (SUBKATEGORISIERUNG: NEGATIVBEISPIEL):
neg_sentence = "der Briefträger beantwortet"
for tree in parser.parse(neg_sentence.split()):
    tree.pretty_print(unicodelines=True)
```

```
               S
       ┌───────┴──────────┐
       NP                 VP
   ┌───┴──────┐           │
  Det         N           V
   │          │           │
  der    Briefträger beantwortet
```

6c)

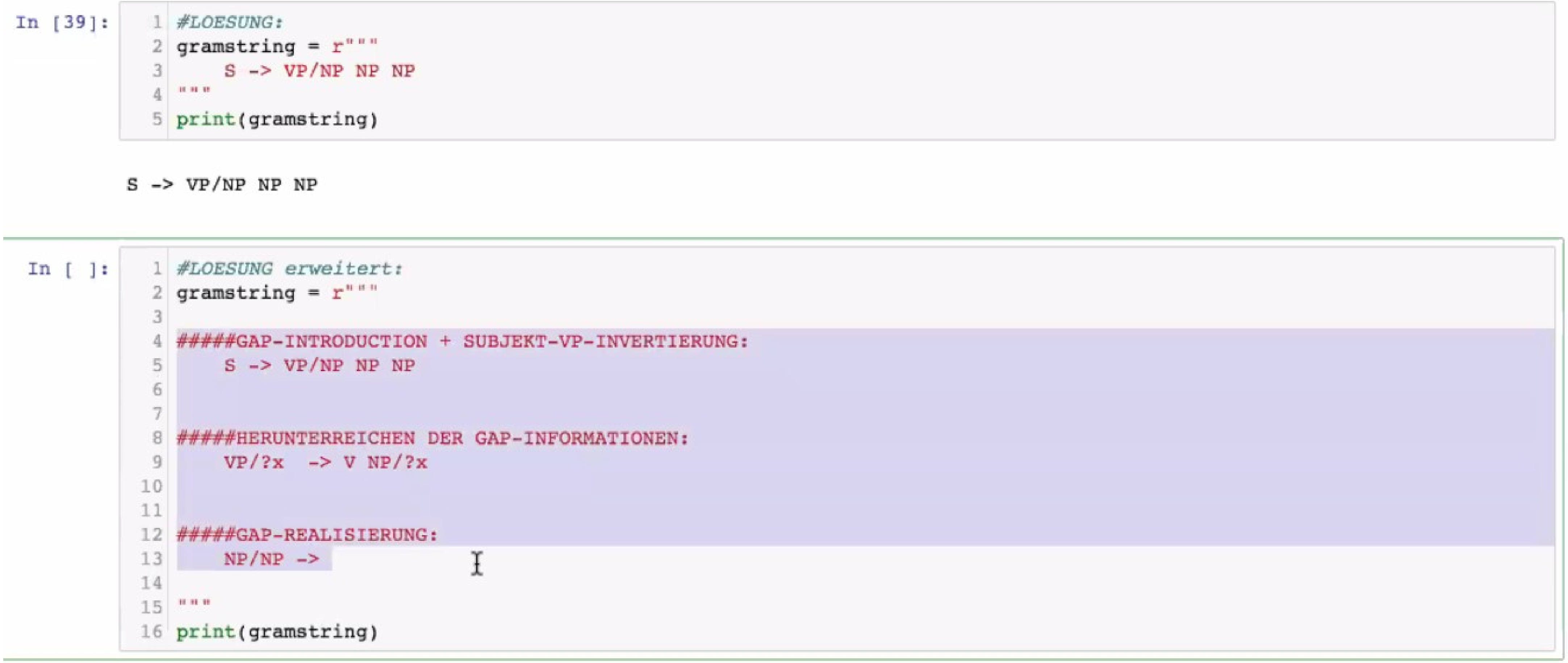


In [22]:

```
#LOESUNG (SUBKATEGORISIERUNG):
gramstring = r"""
% start S

########GRAMMATIK AUS 6a):
    S   -> NP VP
    VP  -> V[SUBCAT="TV"] NP
    VP  -> V[SUBCAT="ITV"]
    NP  -> Det N

    Det -> "der"
    Det -> "den"
    N   -> "Brief"
    N   -> "Briefträger"
    V[SUBCAT="TV"]   -> "beantwortet"
    V[SUBCAT="ITV"]   -> "rennt"

"""

grammar = nltk.grammar.FeatureGrammar.fromstring(gramstring)
parser = nltk.parse.FeatureChartParser(grammar,trace=1)

#NEGATIVBEISPIEL (neg_sentence aus 6b):
for tree in parser.parse(neg_sentence.split()):
    tree = Tree.fromstring(str(tree).replace(",  ",","))
    tree.pretty_print(unicodelines=True)
    #display(tree)
```

```
| der  Brie bean |
```

7

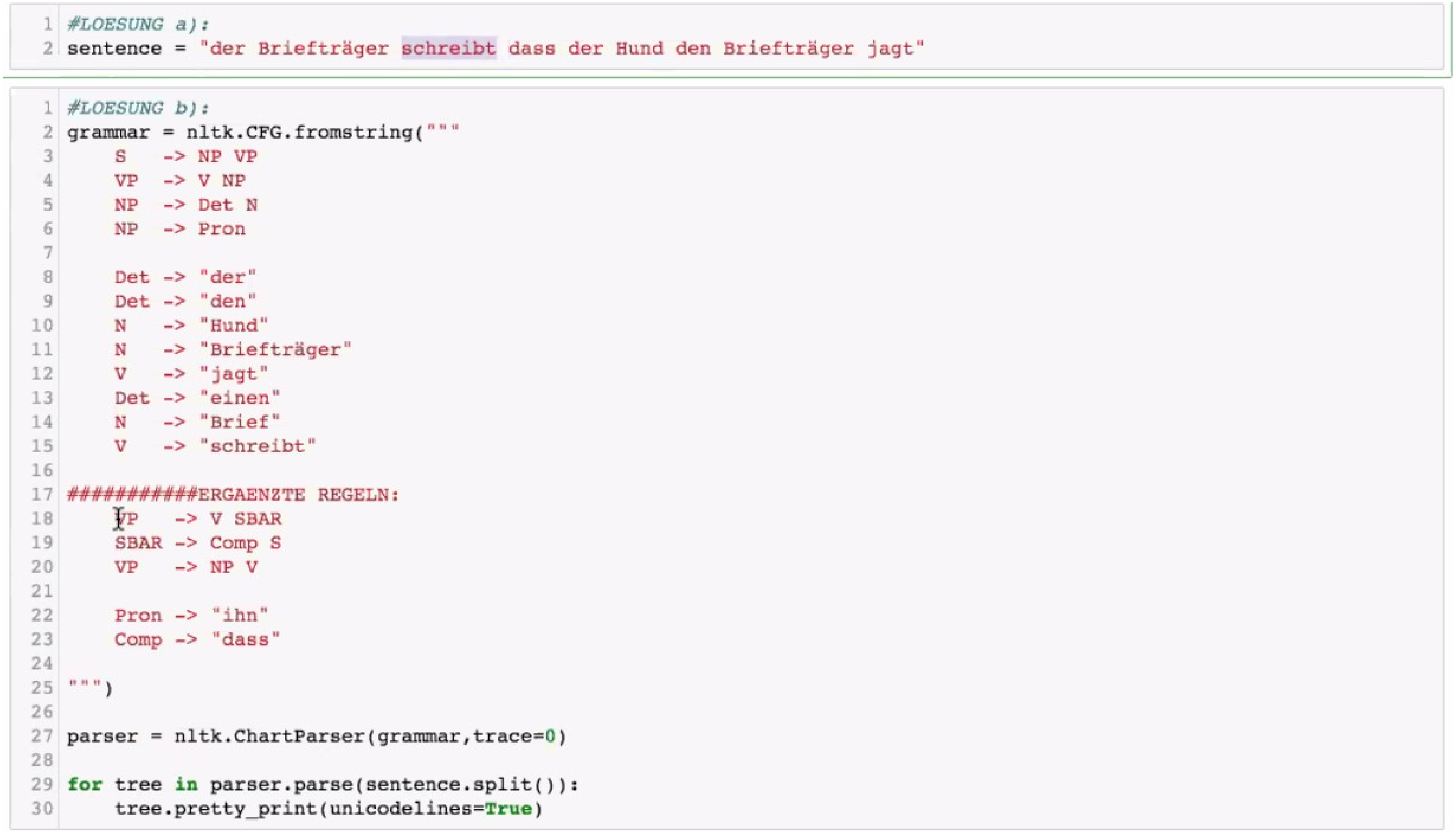


```
#LOESUNG:
gramstring = r"""
    S -> VP/NP NP NP
"""
print(gramstring)
```

```
S -> VP/NP NP NP
```

```
#LOESUNG erweitert:
gramstring = r"""

#####GAP-INTRODUCTION + SUBJEKT-VP-INVERTIERUNG:
    S -> VP/NP NP NP


#####HERUNTERREICHEN DER GAP-INFORMATIONEN:
    VP/?x  -> V NP/?x


#####GAP-REALISIERUNG:
    NP/NP -> 

"""
print(gramstring)
```

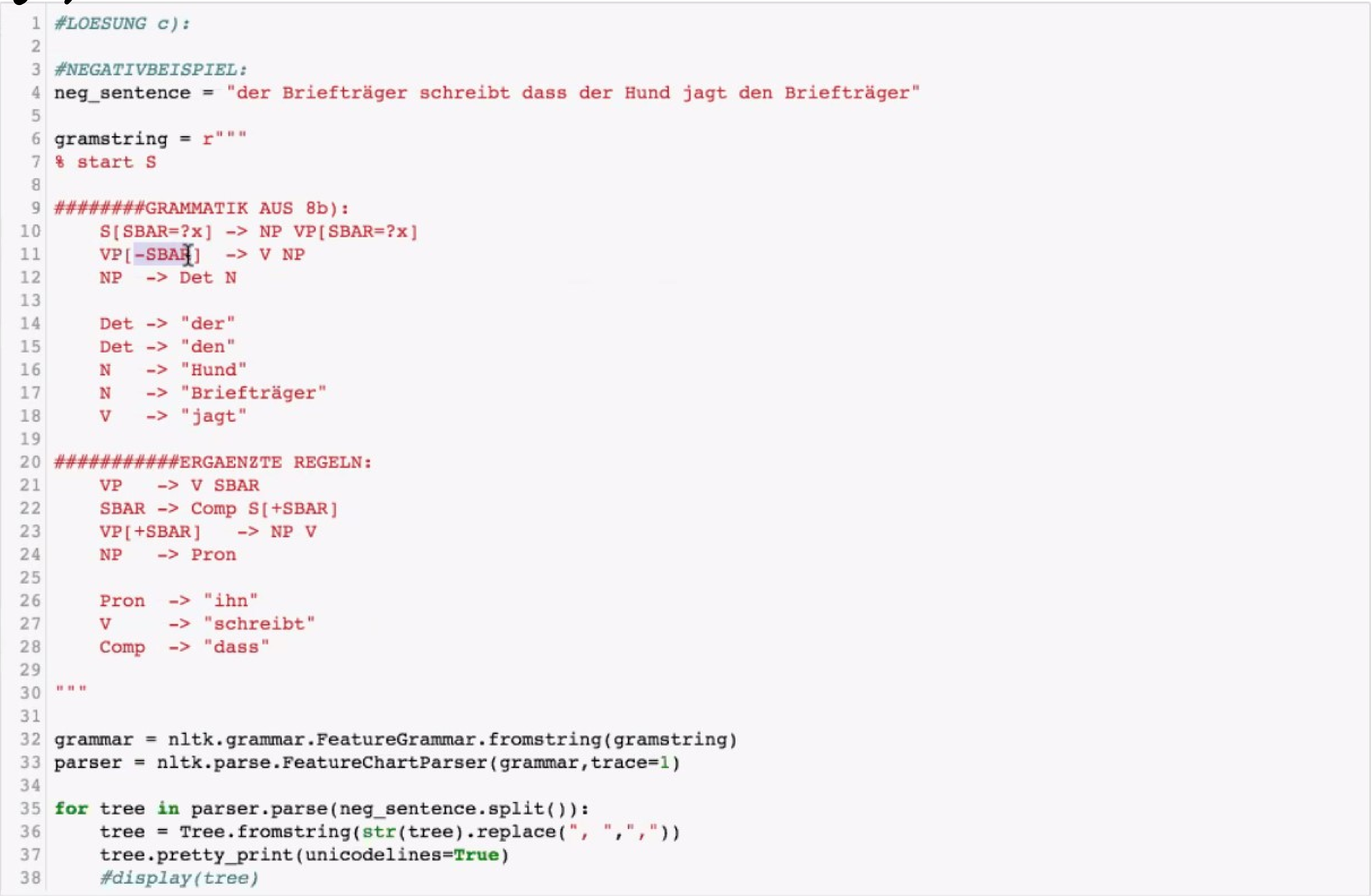
8 a)b)



```
#LOESUNG a):
sentence = "der Briefträger schreibt dass der Hund den Briefträger jagt"
```

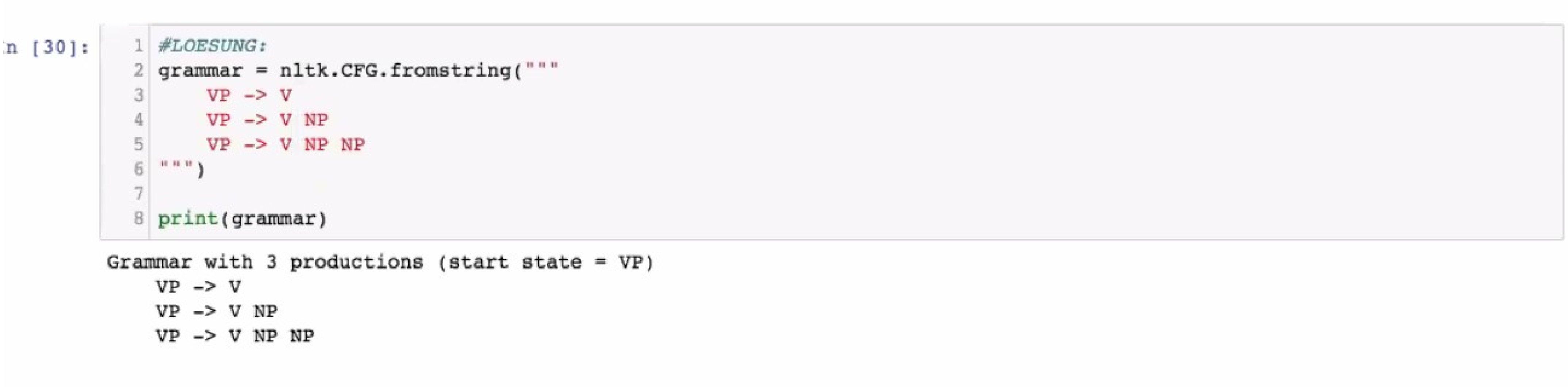
```
#LOESUNG b):
grammar = nltk.CFG.fromstring("""
    S   -> NP VP
    VP  -> V NP
    NP  -> Det N
    NP  -> Pron

    Det -> "der"
    Det -> "den"
    N   -> "Hund"
    N   -> "Briefträger"
    V   -> "jagt"
    Det -> "einen"
    N   -> "Brief"
    V   -> "schreibt"

###########ERGAENZTE REGELN:
    VP   -> V SBAR
    SBAR -> Comp S
    VP   -> NP V

    Pron -> "ihn"
    Comp -> "dass"

""")

parser = nltk.ChartParser(grammar,trace=0)

for tree in parser.parse(sentence.split()):
    tree.pretty_print(unicodelines=True)
```

8c)



```
#LOESUNG c):

#NEGATIVBEISPIEL:
neg_sentence = "der Briefträger schreibt dass der Hund jagt den Briefträger"

gramstring = r"""
% start S

#######GRAMMATIK AUS 8b):
    S[SBAR=?x] -> NP VP[SBAR=?x]
    VP[-SBAR]  -> V NP
    NP  -> Det N

    Det -> "der"
    Det -> "den"
    N   -> "Hund"
    N   -> "Briefträger"
    V   -> "jagt"

##########ERGAENZTE REGELN:
    VP   -> V SBAR
    SBAR -> Comp S[+SBAR]
    VP[+SBAR]   -> NP V
    NP   -> Pron

    Pron  -> "ihn"
    V     -> "schreibt"
    Comp  -> "dass"

"""

grammar = nltk.grammar.FeatureGrammar.fromstring(gramstring)
parser = nltk.parse.FeatureChartParser(grammar,trace=1)

for tree in parser.parse(neg_sentence.split()):
    tree = Tree.fromstring(str(tree).replace(", ",","))
    tree.pretty_print(unicodelines=True)
    #display(tree)
```

9.1

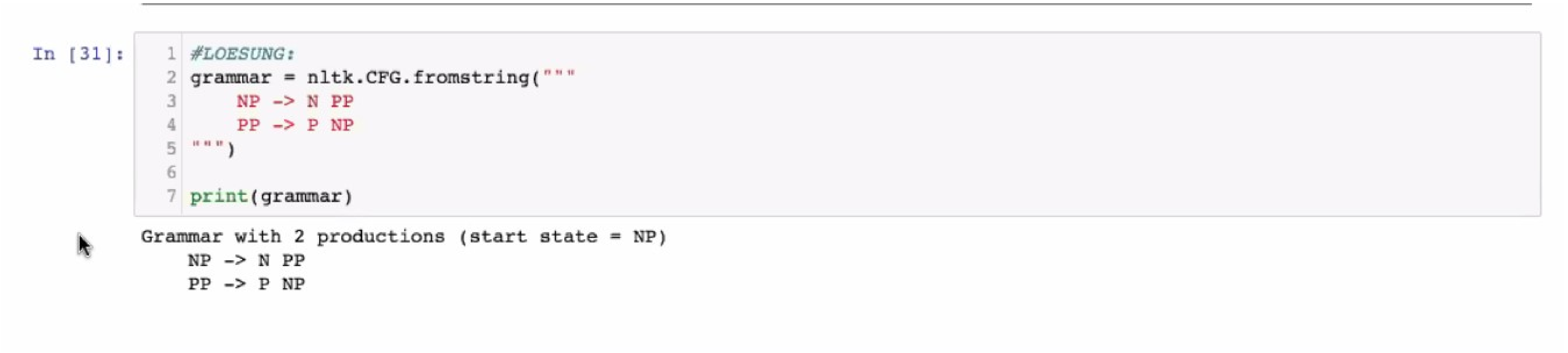


n [30]:

```
#LOESUNG:
grammar = nltk.CFG.fromstring("""
    VP -> V
    VP -> V NP
    VP -> V NP NP
""")

print(grammar)
```

```
Grammar with 3 productions (start state = VP)
    VP -> V
    VP -> V NP
    VP -> V NP NP
```

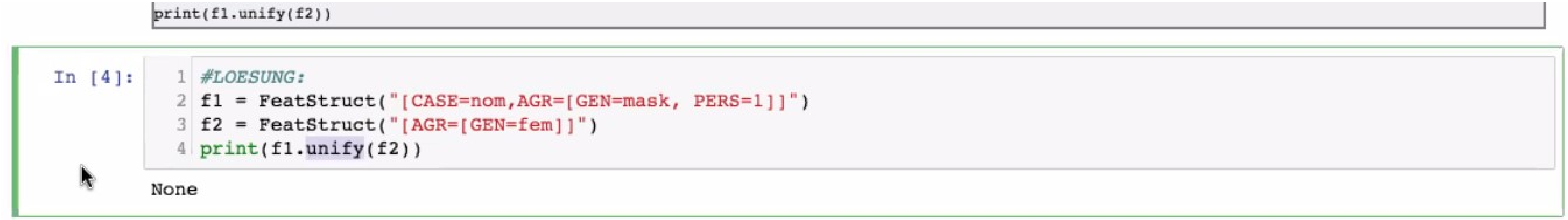
9.2



```
#LOESUNG:
grammar = nltk.CFG.fromstring("""
    NP -> N PP
    PP -> P NP
""")

print(grammar)
```

```
Grammar with 2 productions (start state = NP)
    NP -> N PP
    PP -> P NP
```

10a)



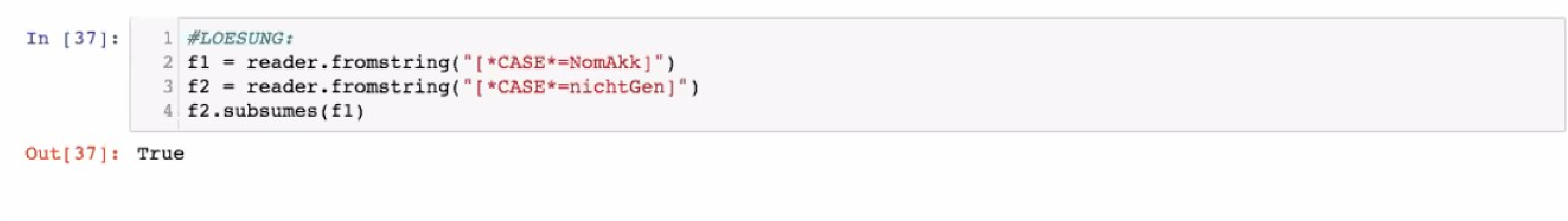
```
print(f1.unify(f2))
```

In [4]:

```
#LOESUNG:
f1 = FeatStruct("[CASE=nom,AGR=[GEN=mask, PERS=1]]")
f2 = FeatStruct("[AGR=[GEN=fem]]")
print(f1.unify(f2))
```

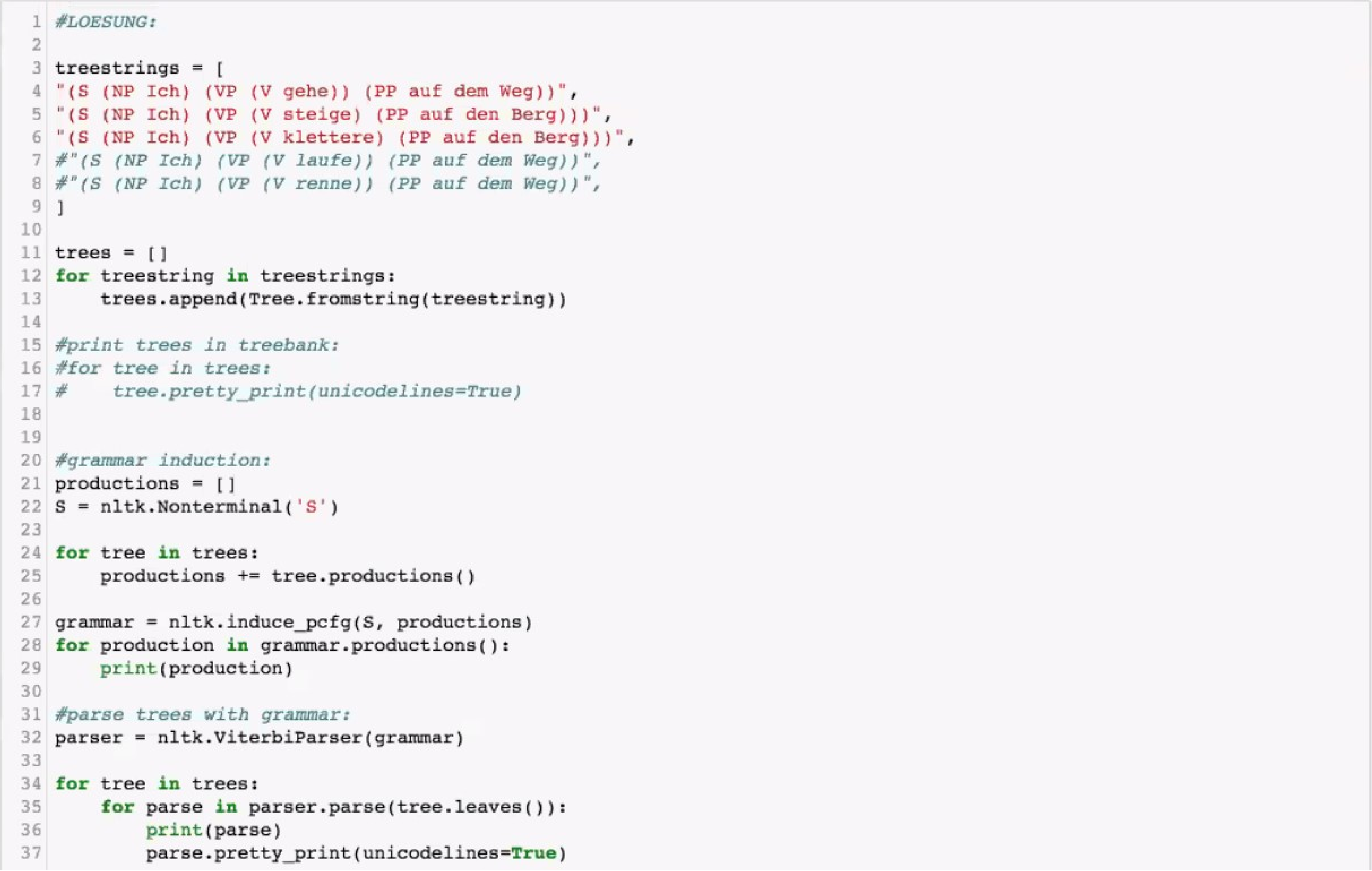
```
None
```

10b)



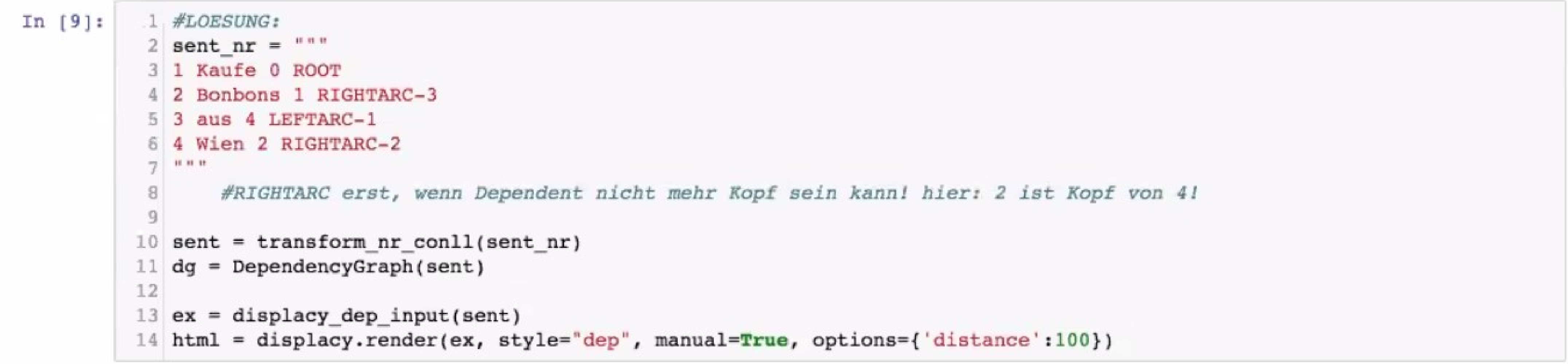
```
In [37]:
#LOESUNG:
f1 = reader.fromstring("[*CASE*=NomAkk]")
f2 = reader.fromstring("[*CASE*=nichtGen]")
f2.subsumes(f1)

Out[37]: True
```

11a)



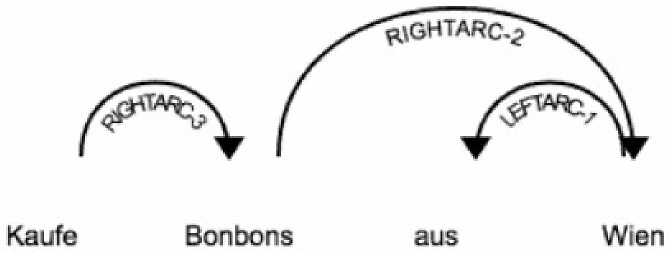
```
#LOESUNG:

treestrings = [
"(S (NP Ich) (VP (V gehe)) (PP auf dem Weg))",
"(S (NP Ich) (VP (V steige) (PP auf den Berg)))",
"(S (NP Ich) (VP (V klettere) (PP auf den Berg)))",
#"(S (NP Ich) (VP (V laufe)) (PP auf dem Weg))",
#"(S (NP Ich) (VP (V renne)) (PP auf dem Weg))",
]

trees = []
for treestring in treestrings:
    trees.append(Tree.fromstring(treestring))

#print trees in treebank:
#for tree in trees:
#    tree.pretty_print(unicodelines=True)


#grammar induction:
productions = []
S = nltk.Nonterminal('S')

for tree in trees:
    productions += tree.productions()

grammar = nltk.induce_pcfg(S, productions)
for production in grammar.productions():
    print(production)

#parse trees with grammar:
parser = nltk.ViterbiParser(grammar)

for tree in trees:
    for parse in parser.parse(tree.leaves()):
        print(parse)
        parse.pretty_print(unicodelines=True)
```

11b)

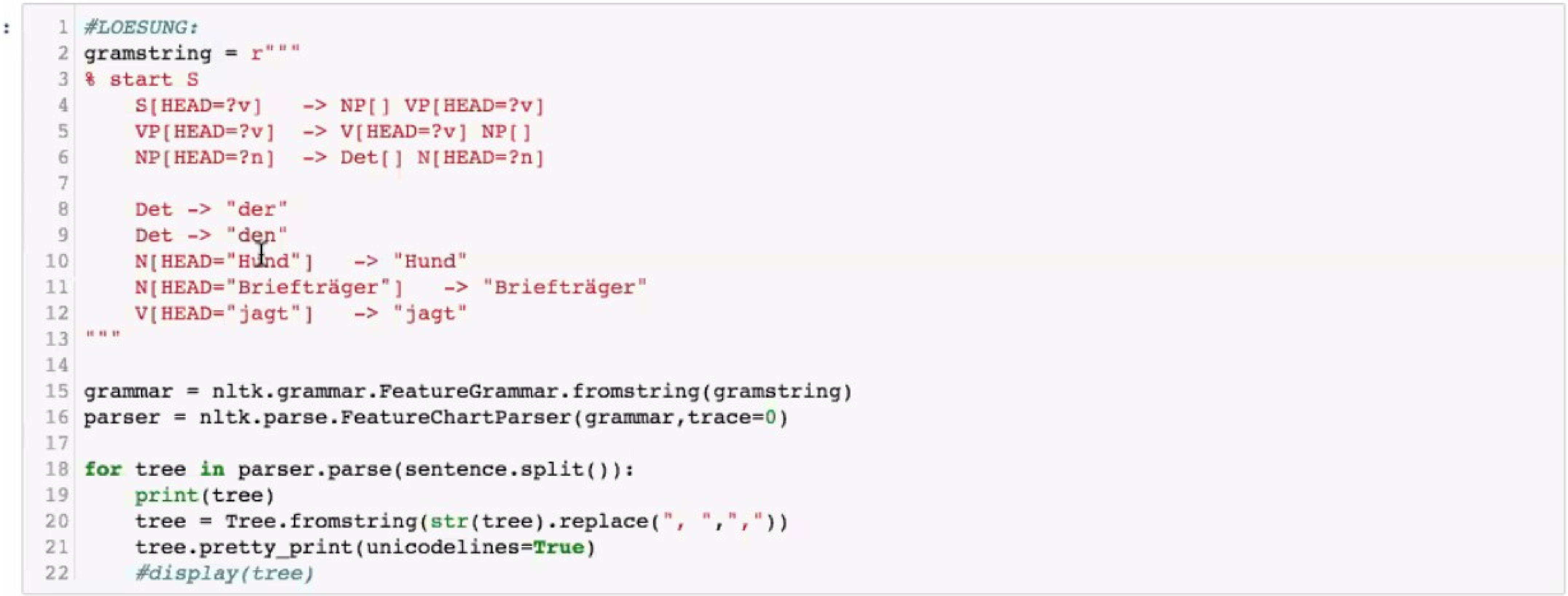


In [9]:

```
#LOESUNG:
sent_nr = """
1 Kaufe 0 ROOT
2 Bonbons 1 RIGHTARC-3
3 aus 4 LEFTARC-1
4 Wien 2 RIGHTARC-2
"""
    #RIGHTARC erst, wenn Dependent nicht mehr Kopf sein kann! hier: 2 ist Kopf von 4!

sent = transform_nr_conll(sent_nr)
dg = DependencyGraph(sent)

ex = displacy_dep_input(sent)
html = displacy.render(ex, style="dep", manual=True, options={'distance':100})
```

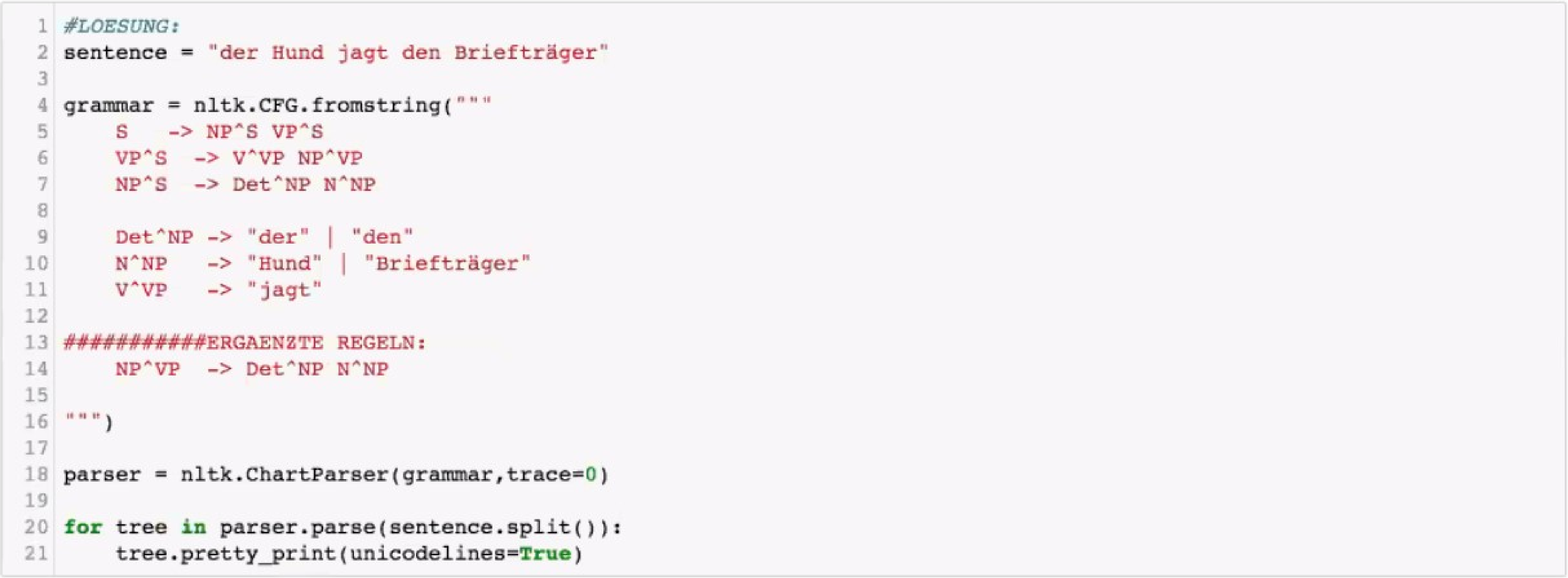


12a)



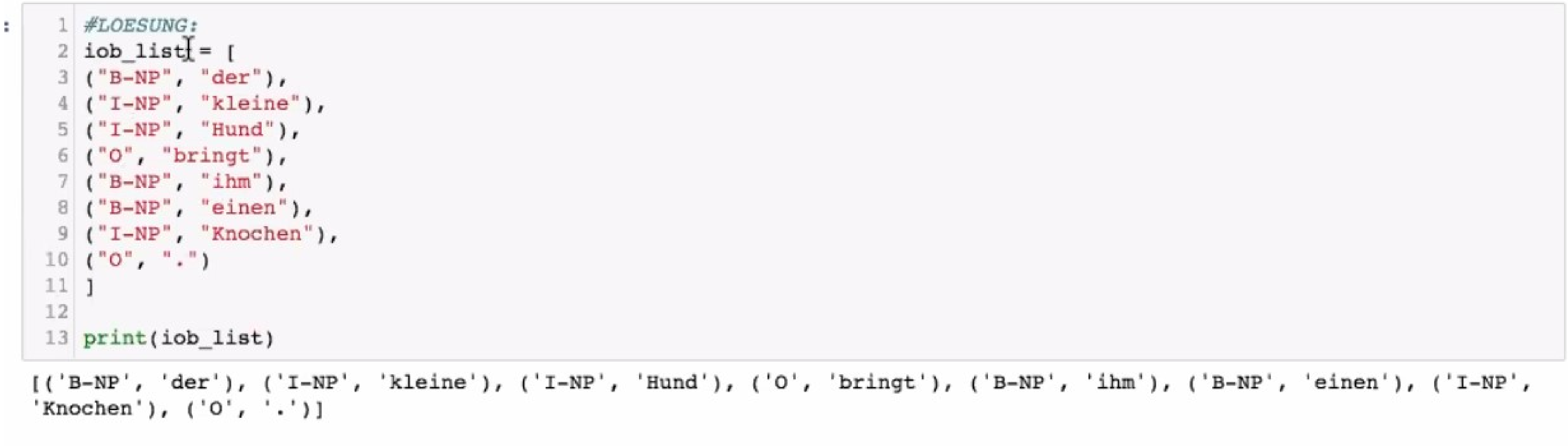
```
#LOESUNG:
gramstring = r"""
% start S
    S[HEAD=?v]   -> NP[] VP[HEAD=?v]
    VP[HEAD=?v]  -> V[HEAD=?v] NP[]
    NP[HEAD=?n]  -> Det[] N[HEAD=?n]

    Det -> "der"
    Det -> "den"
    N[HEAD="Hund"]   -> "Hund"
    N[HEAD="Briefträger"]   -> "Briefträger"
    V[HEAD="jagt"]   -> "jagt"
"""

grammar = nltk.grammar.FeatureGrammar.fromstring(gramstring)
parser = nltk.parse.FeatureChartParser(grammar,trace=0)

for tree in parser.parse(sentence.split()):
    print(tree)
    tree = Tree.fromstring(str(tree).replace(", ",","))
    tree.pretty_print(unicodelines=True)
    #display(tree)
```

12b)



```
#LOESUNG:
sentence = "der Hund jagt den Briefträger"

grammar = nltk.CFG.fromstring("""
    S    -> NP^S VP^S
    VP^S   -> V^VP NP^VP
    NP^S   -> Det^NP N^NP

    Det^NP -> "der" | "den"
    N^NP    -> "Hund" | "Briefträger"
    V^VP    -> "jagt"

###########ERGAENZTE REGELN:
    NP^VP   -> Det^NP N^NP

""")

parser = nltk.ChartParser(grammar,trace=0)

for tree in parser.parse(sentence.split()):
    tree.pretty_print(unicodelines=True)
```

13



```
#LOESUNG:
iob_list = [
("B-NP", "der"),
("I-NP", "kleine"),
("I-NP", "Hund"),
("O", "bringt"),
("B-NP", "ihm"),
("B-NP", "einen"),
("I-NP", "Knochen"),
("O", ".")
]

print(iob_list)
```

```
[('B-NP', 'der'), ('I-NP', 'kleine'), ('I-NP', 'Hund'), ('O', 'bringt'), ('B-NP', 'ihm'), ('B-NP', 'einen'), ('I-NP',
'Knochen'), ('O', '.')]
```