```
import nltk
nltk.download("punkt")
from nltk.tree import Tree
from nltk.tokenize import word_tokenize.
from IPython.display import display
import nltk, re, pprint.
from nltk.tag import pos_tag
from nltk.chunk import ne_chunk
import numpy as npt.
! apt-get install -y xvfb # Install X Virtual Frame Buffer.
import os
os.system('Xvfb :1 -screen 0 1600x1200x16 &') # Create virtual.
    display - with size 1600 x 1200 and 16 bit color. Color can
    be changed to 24 or 8.
os.environ['DISPLAY'] = ':1.0'.
% matplotlib inline
```

Exercise - 1

```
Grammar1 = nltk.CFG.fromstring("""
S -> NP VP | NP VP.
NP -> N | Det N | PRO | NN.
VP -> V NP CP | VP ADVP | V NP.
ADVP -> ADV ADV
CP -> COMP S.
N -> 'Lisa' | 'brother' | 'peanut' | 'butter'
V -> 'told' | 'liked'
```

Natural Language Processing Lab
Lab12. Building and Parsing Context Free Grammars

In this lab, you will create Context Free Grammars for the given sentences and parse these sentences using the grammar you wrote.

Please remember the following points, while writing your grammar.

- In the trees and rules, **use lower-case** ('the', 'he') for all words, even at the beginning of a sentence. The only exceptions are the proper names ('Homer', 'Marge', etc.). This simplifies grammar development and parsing.
- For the same reason, **disregard punctuation and symbols** for this assignment.
- For your reference, all the sentences and their tree drawings used in this assignment can be found on this page. Make sure the trees you build matches the tree representation on it.

EXERCISE-1: Build Grammar and Parser

Your job for this part is to develop a context-free grammar (CFG) and a chart parser (as shown in 8.1.2 Ubiquitous ambiguity using the grammar.

1. Using NLTK's `nltk.CFG.fromstring()` method, build a CFG named `grammar1`. The grammar should cover all of the sentences below and their tree structure as presented on this page. The grammar's start symbol should be `'S'`: make sure that an S rule (ex. S -> NP VP) is the very top rule in your list of rules.

   (s6): the big bully punched the tiny nerdy kid after school
   (s7): he gave the book to his sister
   (s8): he gave the book that I had given him t to his sister
   (s9): Homer and Marge are poor but very happy
   (s10): Homer and his friends from work drank and sang in the bar
   (s11): Lisa told her brother that she liked peanut butter very much

2. Once a grammar is built, you can `print` it. Also, you can extract a set of production rules with the `.productions()` method. Unlike the `.productions()` method called on a `Tree` object, the resulting list should be duplicate-free. As before, each rule in the list is a production rule type. A rule has a left-hand side node (the parent node), which you can get to using the `.lhs()` method; the actual string label for the node can be accessed by calling `.symbol()` on the node object.

```
>>> print(grammar3)
Grammar with 5 productions (start state = S)
    S -> NP VP
    NP -> N
    VP -> V
    N -> 'Homer'
    V -> 'sleeps'

>>> grammar3.productions()
[S -> NP VP, NP -> N, VP -> V, N -> 'Homer', V -> 'sleeps']

>>> last_rule = grammar3.productions()[-1]
>>> last_rule
V -> 'sleeps'

>>> last_rule.is_lexical()
True
```

Comp → 'that'

Det → 'her'

PRO → 'she'

ADV → 'very' | 'much'

S → NP VP

NP → NP CONJ NP | N | NP PP | Det N | N | Det N.

VP → VP PP | VP CONJ VP | V | V

PP → P NP | PNP

N → 'Homer' | 'friends' | 'work' | 'bar'

V → 'drank' | 'sang'

CONJ → 'and' | 'and'

Det → 'his' | 'the'

P → 'from' | 'in'

S → NP VP

NP → NP CONJ NP | N | N

VP → V ADJP

ADJP → ADJP CONJ ADJP | ADJ | ADV ADJ

N → 'Homer' | 'Marge'

V → 'are'

CONJ → 'and' | 'but'

ADJ → 'poor' | 'happy'

ADV → 'very'

S → NP VP | NP AUX VP

NP → 'PRO' | NP CP | Det N | PRO | PRO | PRO | N | Det N

NP → V NP PP | V NP NP

CP → Comp S.

PP → P NP

Det → 'the' | 'his'

```
>>> last_rule.lhs()                    # returns root node object
V

>>> last_rule.lhs().symbol()           # returns node label as a string
'V'
```

3. Explore the rules and answer the following questions.
   a. What is the start state of your grammar?
   b. How many CF rules are in your grammar? Is it 71? (It should be.)
   c. How many of them are lexical?
   d. How many VP rules are there? That is, how many rules have 'VP' on the left-hand side of the rule? That is, how many rules are of the `VP -> ...` form?
   e. How many V rules are there? That is, how many rules have 'V' on the left-hand side of the fule? That is, how many rules are of the `V -> ...` form?
4. Using `grammar1`, build a **chart parser**. (Example shown in 8.1.2 Ubiquitous ambiguity.)
5. Using the parser, parse the sentences `s6` -- `s11`. If your `grammar1` is built correctly to cover all of the sentences, the parser should successfully parse all of them.

PRO → 'he' | 'I' | 'him'

N → 'book' | 'it' | 'sister'

V → 'gave' | 'given'

Comp → 'that'

AUX → 'had'

P → 'to'

S → Np VP

Np → Det ADJ N | Det ADJ ADJ N | N.

VP → V Np | VP PP.

PP → P Np.

Det → 'the' | 'the'

ADJ → 'big' | 'tiny' | 'nasty'

N → 'bully' | 'kid' | 'school'

V → 'punched'

P → 'after'

```
sb_grammar1 = nltk.CFG.fromstring("""
S -> NP VP.
NP -> Det ADJ N | Det ADJ ADJ N | N.
VP -> VNP | VP PP.
PP -> P NP.
Det -> 'the' | 'the'
ADJ -> 'big' | 'tiny' | 'nerdy'
N -> 'bully' | 'kid' | 'school'
V -> 'punched'
P -> 'after'
""")

sentence6 = word_tokenize("the big bully punched the tiny kid after. school")

parser = nltk.ChartParser(sb_grammar1)
for tree in parser.parse(sentence6):
    print(tree)

np6 = nltk.Tree.fromstring('(S (NP (Det the) (ADJ big) (N bully)) (VP (VP V_punched) (NP (Det the) (ADJ tiny) (ADJ nerdy) (N kid)) (PP (P after) (NP (N school))))))')

display(np1)

sf_grammar1 = nltk.CFG.fromstring("""
S -> NP VP.
NP -> PRO | Det N | Det N.
VP -> V NP PP
PP -> P NP
Det -> 'the' | 'his'
PRO -> 'he'
N -> 'book' | 'sister'
V -> 'gave'
P -> 'to'
""")
```

NOTES

```
sentence7 = word_tokenize("he gave the book to his sister")
parser = nltk.chartParser(cf_grammar1)
for tr in parser.parse(sentence7):
    print(tr)
np7 = nltk.Tree.fromstring('(S(NP(PRO he))(VP(V gave)
    (NP(Det the)(N book))(PP(P to)
    (NP(Det his)(N sister)))))').
display(np7)
CSS_grammar1 = nltk.CFG.fromstring("""
S -> NP VP | NP AUX VP.
NP -> PRO | NP CP | Det N | PRO | PRO PRO | N | Det N.
VP -> V NP PP | V NP NP.
CP -> COMP S.
PP -> P NP.
Det -> 'the' | 'his'
PRO -> 'he' | 'I' | 'him'
N -> 'book' | 'it' | 'sister'
V -> 'gave' | 'given'
COMP -> 'that'
AUX -> 'had'
P -> 'to'
""")
sentence8 = word_tokenize("he gave the book that I had
    given him to his sister")
parser = nltk.chartparser(CSS_grammar1)
for tr in parser.parse(sentence8):
    print(tr)
```

```
nps = nltk.Tree.fromstring('(S(NP(PRO he))(VP(V gave)
(NP(NP(Det the)(N book))(CP(COMP that)(S(NP
(PRO I))(AUX had)(VP(V given)(NP(PRO him))(NP(N it))))
(PP(P to)(NP(Det his)(N sister))))))')
display(nps)
```

(a9)

```
a9_grammar1 = nltk.CFG.fromstring("""
S -> NP VP
N -> NP CONJ NP | N | N
VP -> V ADJP
ADJP -> ADJP CONJ ADJP | ADJ | ADV ADJ
N -> 'Homer' | 'Marge'
V -> 'are'
CONJ -> 'and' | 'but'
ADJ -> 'poor' | 'happy'
ADV -> 'very'
""")
sentence9 = word_tokenize("Homer and Marge are poor.
but very happy")
parser = nltk.ChartParser(a9_grammar1)
for p in parser.parse(sentence9):
    print(p)

np9 = nltk.Tree.fromstring('(S(NP(NP(N Homer))
(CONJ and)(NP(N Marge)))(VP(V are)(ADJP
(ADJP(ADJ poor))(CONJ but)(ADJP(ADV very)
(ADJ happy)))))')
display(np9)
```

```
s_10 - grammar1 = nltk.CFG.fromstring("""
S -> NP VP.
NP -> NP CONJ NP | N | NP PP | Det N | N | Det N.
VP -> VP PP | VP CONJ VP | V | V.
```

NOTES

```
PP -> P NP | PNP.
N -> 'Homer' | 'friends' | 'work' | 'bar'
V -> 'drank' | 'sang'
ConJ -> 'and' | 'and'.
Det -> 'his' | 'the'
P -> 'from' | 'in'
""")

sentence10 = word_tokenize("Homer and his friends from work drank and sang in the bar")

parser = nltk.ChartParser(S10_grammar1)

for t in parser.parse(sentence10):
    print(t)

np10 = nltk.Tree.fromstring('(S(NP(NP(N Homer))
(ConJ and)(NP(NP(Det his)(N friends))(PP(P from
(V sang))(PP(P in)(NP(Det the)(N bar))))))')

display(np10)

S1_grammar1 = nltk.CFG.fromstring("""
S -> NP VP | NP VP.
NP -> N | Det N | PRO | V NP.
VP -> V NP CP | VP ADVP | V NP.
ADVP -> ADV ADV.
CP -> COMP S.
N -> 'Bob' | 'brother' | 'peanut' | 'butter'
V -> 'told' | 'liked'
COMP -> 'that'
Det -> 'her'.
PRO -> 'she'
ADV -> 'very' | 'much'
""")
print(t)
```