```
import nltk, re, pprint.
from nltk.tree import Tree
from nltk.tokenize import word_tokenize
from nltk.tag import pos_tag
from nltk.chunk import re_chunk.
import numpy as np.
```

Exe:1

```
np = nltk.Tree.fromstring('(NP(N Marge))')
np.pretty_print()
vp = nltk.Tree.fromstring('(VP (V make) (NP (DET a)
                                (N ham) (N sandwich)))')
vp.pretty_print()
```

Exe:2

```
s1 = nltk.Tree.fromstring('(S(NP(N Marge))
                            (AUX will) (VP(V make) (NP
                            (DET a)
s1.pretty_print()
s2 = nltk.Tree.fromstring('(S(AUX will) (NP(N Marge))
     (VP(V make) (NP (DET a)
s2.pretty.print()
```

Exe:4

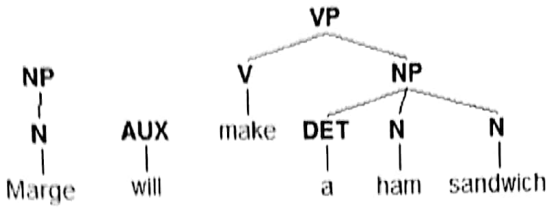
```
s3 = nltk.Tree.fromstring ('(S (NP(N Homer)) (VP(V ate)
     (NP(DET the) (N donut))
s3.pretty_print()
```

Natural Language Processing Lab
Lab11. Building Parse Trees

In this lab, you will build parse trees for the given sentences.

EXERCISE-1

Build the following three tree objects as `np`, `aux`, and `vp`.



EXERCISE-2

Create a parse tree for the phrase ***old men and women.*** Is it **well formed sentence** or ambiguous sentence?.

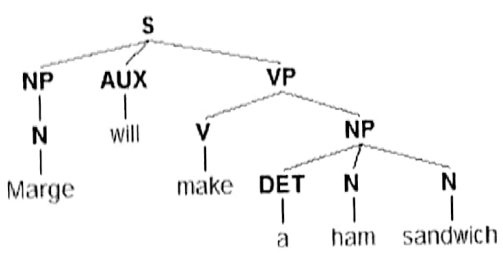
Steps:

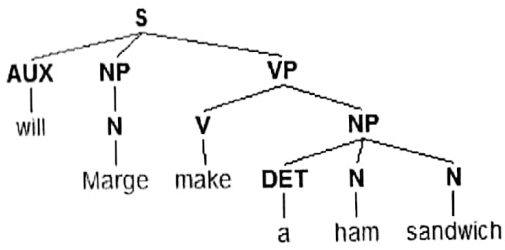
1. Define the grammar (use fromstring() method)
2. Create sentence (as a list of words)
3. Create chart parser
4. Parse and print tree(s)

EXERCISE-3

Using them, build two tree objects, named `s1` and `s2`, for the following sentences. The trees should look exactly like the ones shown below

(s1) Marge will make a ham sandwich
(s2) will Marge make a ham sandwich





EXERCISE-4

Build a tree object named `s3` for the following sentence, using its full-sentence string representation.

(s3) Homer ate the donut on the table

Exe:5

```
S4 = nltk.Tree.fromstring('(S (NP (DET my) (ADJ old)
                (N cat)) (VP (VP (V died)) (PP (P
S4.pretty_print()

S5 = nltk.Tree.fromstring('(S (NP (N children)) (Aux are)
                (VP (VP (V play)) (PP (P in
S5.pretty_print()
```

Exe:6

```
print(VP)
VP_rules = VP.productions()
VP_rules
VP_rules[0]
VP_rules[1]
VP_rules[0].is_lexical()
VP_rules[0].is_lexical()
```

Explore the CF rules of S5

```
print(S5)
S5_rules = S5.productions()
S5_rules
print("How many CF values are used in S5", len(S5.rules))
x = npt.array(S5_rules)
print("How many unique CF rules are used in S5", len(npt.unique(x)))
```

EXERCISE-5

Build tree objects named `s4` and `s5` for the following sentences.

(s4) my old cat died on Tuesday
(s5) children must play in the park with their friends

EXERCISE-6

Once a tree is built, you can extract a list of **context-free rules**, generally called **production rules**, from it using the `.productions()` method. Each CF rule in the list is either lexical, i.e, contains a lexical word on its right-hand side, or not:

```
>>> print(vp)
(VP (V ate) (NP (DET the) (N donut)))
>>> vp_rules = vp.productions()       # list of all CF rules used in the tree
>>> vp_rules
[VP -> V NP, V -> 'ate', NP -> DET N, DET -> 'the', N -> 'donut']
>>> vp_rules[0]
VP -> V NP
>>> vp_rules[1]
V -> 'ate'
>>> vp_rules[0].is_lexical()    # VP -> V NP is not a lexical rule
False
>>> vp_rules[1].is_lexical()    # V -> 'ate' is a lexical rule
True
```

Explore the CF rules of `s5`. Include in your script the answers to the following:

a. How many CF rules are used in `s5`?
b. How many *unique* CF rules are used in `s5`?
c. How many of them are lexical?

c. How many of them are lexical?

```
n=0
for x in s5.productions():
    if x.is_lexical():
        n=n+1
print("How many of them are lexical?", n)
```