```
225229124
```

Exercise-1

In [1]:

```
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 ne_chunk
import numpy as npt
```

In [2]:

```
np= nltk.Tree.fromstring('(NP (N Marge))')
np.pretty_print()
```

```
  NP
  |
  N
  |
Marge
```

In [3]:

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

```
          VP
  ________|___
 |            NP
 |     _______|_____
 V    DET      N     N
 |    |       |     |
make  a      ham sandwich
```

In [4]:

```
aux= nltk.Tree.fromstring('(AUX will)')
aux.pretty_print()
```

```
AUX
 |
will
```

Exercise 2

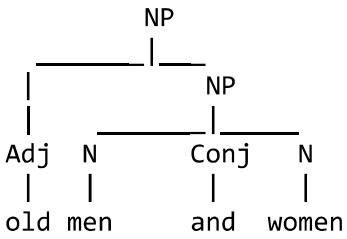
Create a parse tree for the phrase old men and women. Is it well formed sentence or ambiguous sentence?. Steps:

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)

In [5]:

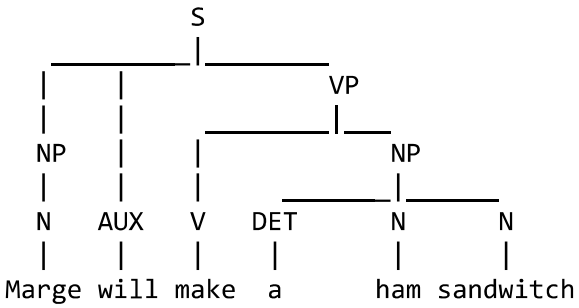
```
tree = nltk.Tree.fromstring('(NP (Adj old) (NP (N men) (Conj and) (N women)))')
tree.pretty_print()
```



Exercise 3

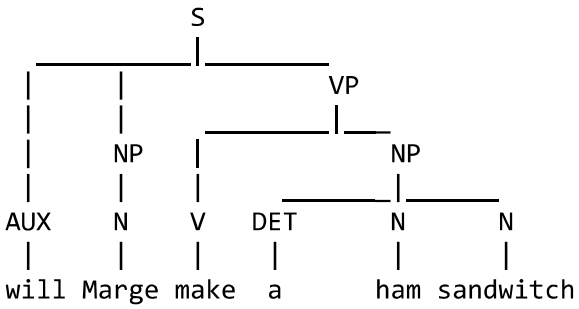
In [6]:

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



In [7]:

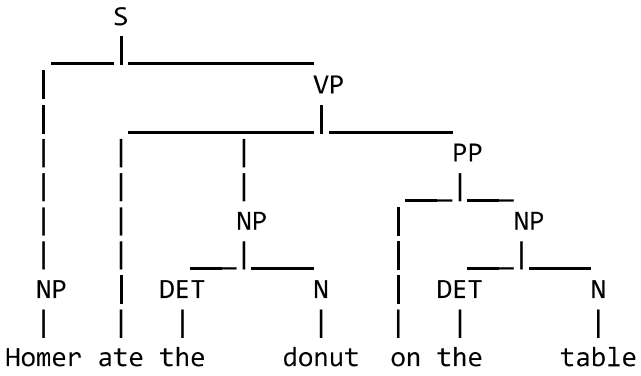
```
s2= nltk.Tree.fromstring('(S (AUX will) (NP (N Marge)) (VP (V make) (NP (DET a) (N ham)
s2.pretty_print()
```



Exercise-4

In [8]:

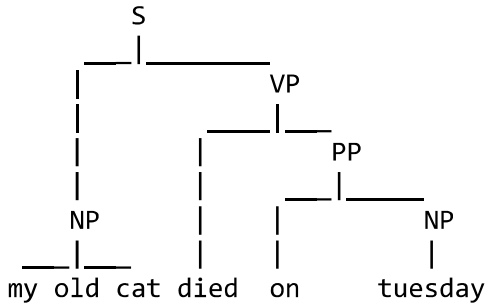
```
s3= nltk.Tree.fromstring('(S (NP Homer) (VP ate (NP (DET the) (N donut)) (PP on (NP (DET
s3.pretty_print()
```



Exercise-5

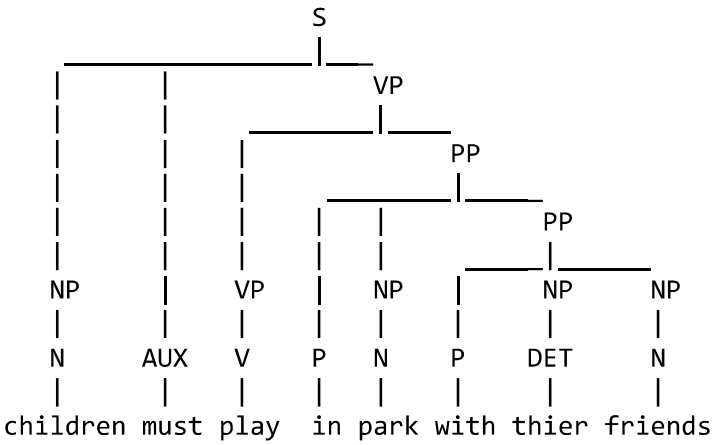
In [9]:

```
s4= nltk.Tree.fromstring('(S (NP my old cat) (VP died (PP on (NP tuesday))))')
s4.pretty_print()
```



In [10]:

```
s5= nltk.Tree.fromstring('(S (NP (N children)) (AUX must) (VP (VP (V play)) (PP (P in) (
s5.pretty_print()
```



Exercise 6

In [11]:

```
print(vp)
```

```
(VP (V make) (NP (DET a) (N ham) (N sandwich)))
```

In [12]:

```
vp_rules= vp.productions() # list of all CF rules used in the tree
vp_rules
```

Out[12]:

```
[VP -> V NP,
 V -> 'make',
 NP -> DET N N,
 DET -> 'a',
 N -> 'ham',
 N -> 'sandwich']
```

In [13]:

```
vp_rules[0]
```

Out[13]:

```
VP -> V NP
```

In [14]:

```
vp_rules[1]
```

Out[14]:

```
V -> 'make'
```

In [15]:

```
vp_rules[0].is_lexical()
```

Out[15]:

```
False
```

In [16]:

```
vp_rules[1].is_lexical()
```

Out[16]:

```
True
```

Explore the CF rules of s5

In [17]:

```
print(s5)
```

```
(S
  (NP (N children))
  (AUX must)
  (VP
    (VP (V play))
    (PP
      (P in)
      (NP (N park))
      (PP (P with) (NP (DET thier)) (NP (N friends))))))
```

In [18]:

```
s5_rules= s5.productions()
s5_rules
```

Out[18]:

```
[S -> NP AUX VP,
 NP -> N,
 N -> 'children',
 AUX -> 'must',
 VP -> VP PP,
 VP -> V,
 V -> 'play',
 PP -> P NP PP,
 P -> 'in',
 NP -> N,
 N -> 'park',
 PP -> P NP NP,
 P -> 'with',
 NP -> DET,
 DET -> 'thier',
 NP -> N,
 N -> 'friends']
```

In [19]:

```
print("How many CF values are used in s5 ",len(s5_rules))
```

```
How many CF values are used in s5  17
```

In [20]:

```
x= npt.array(s5_rules)
print("How many unique CF rules are used in s5 ",len(npt.unique(x)))
```

```
How many unique CF rules are used in s5  15
```

In [21]:

```
n= 0
for x in s5_rules:
    if x.is_lexical():
        n= n+1
print("How many of them are lexical? ",n)
```

```
How many of them are lexical?  8
```