Lab1-Assignment

This notebook describes the assignment for Lab 1 of the text mining course.

Points: each exercise is prefixed with the number of points you can obtain for the exercise.

We assume you have worked through the following notebooks:

- **Lab1.1-introduction**
- **Lab1.2-introduction-to-NLTK**
- **Lab1.3-introduction-to-spaCy**

In this assignment, you will process an English text (Lab1-apple-samsung-example.txt) with both NLTK and spaCy and discuss the similarities and differences.

Credits

The notebooks in this block have been originally created by Marten Postma. Adaptations were made by Filip Ilievski.

Tip: how to read a file from disk

Let's open the file Lab1-apple-samsung-example.txt from disk.

In [117]:

```
from pathlib import Path
```

In [118]:

```
cur_dir = Path().resolve() # this should provide you with the folder in which this noteb
ook is placed
path_to_file = Path.joinpath(cur_dir, 'Lab1-apple-samsung-example.txt')
print(path_to_file)
print('does path exist? ->', Path.exists(path_to_file))
```

```
/Users/robertostoica/Desktop/University_files_(year_3)_extra/period_4/Text Mining for AI/
Assignments/Lab_Assignments/Lab_1/Lab1-apple-samsung-example.txt
does path exist? -> True
```

If the output from the code cell above states that does path exist? -> False , please check that the file Lab1-apple-samsung-example.txt is in the same directory as this notebook.

In [119]:

```
with open(path_to_file) as infile:
    text = infile.read()

print('number of characters', len(text))
```

```
number of characters 1139
```

[total points: 4] Exercise 1: NLTK

In this exercise, we use NLTK to apply Part-of-speech (POS) tagging , Named Entity Recognition (NER), and Constituency parsing. The following code snippet already performs sentence splitting and tokenization.

Constituency parsing: The following code snippet already performs sentence splitting and tokenization.

In [120]:

```
import nltk
from nltk.tokenize import sent_tokenize
from nltk import word_tokenize
```

In [121]:

```
sentences_nltk = sent_tokenize(text)
```

In [122]:

```
tokens_per_sentence = []
for sentence_nltk in sentences_nltk:
    sent_tokens = word_tokenize(sentence_nltk)
    tokens_per_sentence.append(sent_tokens)
```

We will use lists to keep track of the output of the NLP tasks. We can hence inspect the output for each task using the index of the sentence.

In [123]:

```
sent_id = 1
print('SENTENCE', sentences_nltk[sent_id])
print('TOKENS', tokens_per_sentence[sent_id])
```

```
SENTENCE The six phones and tablets affected are the Galaxy S III, running the new Jelly
Bean system, the Galaxy Tab 8.9 Wifi tablet, the Galaxy Tab 2 10.1, Galaxy Rugby Pro and
Galaxy S III mini.
TOKENS ['The', 'six', 'phones', 'and', 'tablets', 'affected', 'are', 'the', 'Galaxy', 'S'
, 'III', ',', 'running', 'the', 'new', 'Jelly', 'Bean', 'system', ',', 'the', 'Galaxy', '
Tab', '8.9', 'Wifi', 'tablet', ',', 'the', 'Galaxy', 'Tab', '2', '10.1', ',', 'Galaxy', '
Rugby', 'Pro', 'and', 'Galaxy', 'S', 'III', 'mini', '.']
```

[point: 1] Exercise 1a: Part-of-speech (POS) tagging

Use `nltk.pos_tag` to perform part-of-speech tagging on each sentence.

Use `print` to show the output in the notebook (and hence also in the exported PDF!).

In [124]:

```
pos_tags_per_sentence = []
for tokens in tokens_per_sentence:
    tagged = nltk.pos_tag(tokens)
    pos_tags_per_sentence.append(tagged)
```

In [125]:

```
print(pos_tags_per_sentence)
```

```
[[('https', 'NN'), (':', ':'), ('//www.telegraph.co.uk/technology/apple/9702716/Apple-Sam
sung-lawsuit-six-more-products-under-scrutiny.html', 'JJ'), ('Documents', 'NNS'), ('filed
', 'VBN'), ('to', 'TO'), ('the', 'DT'), ('San', 'NNP'), ('Jose', 'NNP'), ('federal', 'JJ'
), ('court', 'NN'), ('in', 'IN'), ('California', 'NNP'), ('on', 'IN'), ('November', 'NNP'
), ('23', 'CD'), ('list', 'NN'), ('six', 'CD'), ('Samsung', 'NNP'), ('products', 'NNS'),
('running', 'VBG'), ('the', 'DT'), ('``', '``'), ('Jelly', 'RB'), ('Bean', 'NNP'), ("''",
"''"), ('and', 'CC'), ('``', '``'), ('Ice', 'NNP'), ('Cream', 'NNP'), ('Sandwich', 'NNP')
, ("''", "''"), ('operating', 'VBG'), ('systems', 'NNS'), (',', ','), ('which', 'WDT'), (
'Apple', 'NNP'), ('claims', 'VBZ'), ('infringe', 'VB'), ('its', 'PRP$'), ('patents', 'NNS
'), ('.', '.')], [('The', 'DT'), ('six', 'CD'), ('phones', 'NNS'), ('and', 'CC'), ('table
ts', 'NNS'), ('affected', 'VBN'), ('are', 'VBP'), ('the', 'DT'), ('Galaxy', 'NNP'), ('S',
'NNP'), ('III', 'NNP'), (',', ','), ('running', 'VBG'), ('the', 'DT'), ('new', 'JJ'), ('J
elly', 'NNP'), ('Bean', 'NNP'), ('system', 'NN'), (',', ','), ('the', 'DT'), ('Galaxy', '
NNP'), ('Tab', 'NNP'), ('8.9', 'CD'), ('Wifi', 'NNP'), ('tablet', 'NN'), (',', ','), ('th
```

```
NNP'), ('Tab', 'NNP'), ('8.9', 'CD'), ('Wifi', 'NNP'), ('tablet', 'NN'), (',', ','), ('th
e', 'DT'), ('Galaxy', 'NNP'), ('Tab', 'NNP'), ('2', 'CD'), ('10.1', 'CD'), (',', ','), ('
Galaxy', 'NNP'), ('Rugby', 'NNP'), ('Pro', 'NNP'), ('and', 'CC'), ('Galaxy', 'NNP'), ('S'
, 'NNP'), ('III', 'NNP'), ('mini', 'NN'), ('.', '.')], [('Apple', 'NNP'), ('stated', 'VBD
'), ('it', 'PRP'), ('had', 'VBD'), ('^', 'NNP'), ('acted', 'VBD'), ('quickly', 'RB'), ('a
nd', 'CC'), ('diligently', 'RB'), ("''", "''"), ('in', 'IN'), ('order', 'NN'), ('to', 'TO
'), ('``', '``'), ('determine', 'VB'), ('that', 'IN'), ('these', 'DT'), ('newly', 'RB'),
('released', 'VBN'), ('products', 'NNS'), ('do', 'VBP'), ('infringe', 'VB'), ('many', 'JJ
'), ('of', 'IN'), ('the', 'DT'), ('same', 'JJ'), ('claims', 'NNS'), ('already', 'RB'), ('
asserted', 'VBN'), ('by', 'IN'), ('Apple', 'NNP'), ('.', '.'), ("''", "''")], [('In', 'IN
'), ('August', 'NNP'), (',', ','), ('Samsung', 'NNP'), ('lost', 'VBD'), ('a', 'DT'), ('US
', 'NNP'), ('patent', 'NN'), ('case', 'NN'), ('to', 'TO'), ('Apple', 'NNP'), ('and', 'CC'
), ('was', 'VBD'), ('ordered', 'VBN'), ('to', 'TO'), ('pay', 'VB'), ('its', 'PRP$'), ('ri
val', 'JJ'), ('$', '$'), ('1.05bn', 'CD'), ('(', '('), ('£0.66bn', 'NN'), (')', ')'), ('i
n', 'IN'), ('damages', 'NNS'), ('for', 'IN'), ('copying', 'VBG'), ('features', 'NNS'), ('
of', 'IN'), ('the', 'DT'), ('iPad', 'NN'), ('and', 'CC'), ('iPhone', 'NN'), ('in', 'IN'),
('its', 'PRP$'), ('Galaxy', 'NNP'), ('range', 'NN'), ('of', 'IN'), ('devices', 'NNS'), ('
.', '.')], [('Samsung', 'NNP'), (',', ','), ('which', 'WDT'), ('is', 'VBZ'), ('the', 'DT'
), ('world', 'NN'), ("'s", 'POS'), ('top', 'JJ'), ('mobile', 'NN'), ('phone', 'NN'), ('ma
ker', 'NN'), (',', ','), ('is', 'VBZ'), ('appealing', 'VBG'), ('the', 'DT'), ('ruling', '
NN'), ('.', '.')], [('A', 'DT'), ('similar', 'JJ'), ('case', 'NN'), ('in', 'IN'), ('the',
'DT'), ('UK', 'NNP'), ('found', 'VBD'), ('in', 'IN'), ('Samsung', 'NNP'), ("'s", 'POS'),
('favour', 'NN'), ('and', 'CC'), ('ordered', 'VBD'), ('Apple', 'NNP'), ('to', 'TO'), ('pu
blish', 'VB'), ('an', 'DT'), ('apology', 'NN'), ('making', 'VBG'), ('clear', 'JJ'), ('tha
t', 'IN'), ('the', 'DT'), ('South', 'JJ'), ('Korean', 'JJ'), ('firm', 'NN'), ('had', 'VBD
'), ('not', 'RB'), ('copied', 'VBN'), ('its', 'PRP$'), ('iPad', 'NN'), ('when', 'WRB'), (
'designing', 'VBG'), ('its', 'PRP$'), ('own', 'JJ'), ('devices', 'NNS'), ('.', '.')]]
```

[point: 1] Exercise 1b: Named Entity Recognition (NER)

Use `nltk.chunk.ne_chunk` to perform Named Entity Recognition (NER) on each sentence.

Use `print` to show the output in the notebook (and hence also in the exported PDF!).

In [126]:

```
ner_tags_per_sentence = []

for sentence in sentences_nltk:

    tokens = nltk.word_tokenize(sentence)
    tokens_pos_tagged = nltk.pos_tag(tokens)
    tokens_pos_tagged_and_named_entities = nltk.chunk.ne_chunk(tokens_pos_tagged)
    ner_tags_per_sentence.append(tokens_pos_tagged_and_named_entities)
```

In [127]:

```
print(ner_tags_per_sentence)
```

```
[Tree('S', [('https', 'NN'), (':', ':'), ('//www.telegraph.co.uk/technology/apple/9702716
/Apple-Samsung-lawsuit-six-more-products-under-scrutiny.html', 'JJ'), ('Documents', 'NNS'
), ('filed', 'VBN'), ('to', 'TO'), ('the', 'DT'), Tree('ORGANIZATION', [('San', 'NNP'), (
'Jose', 'NNP')]), ('federal', 'JJ'), ('court', 'NN'), ('in', 'IN'), Tree('GPE', [('Califo
rnia', 'NNP')]), ('on', 'IN'), ('November', 'NNP'), ('23', 'CD'), ('list', 'NN'), ('six',
'CD'), Tree('ORGANIZATION', [('Samsung', 'NNP')]), ('products', 'NNS'), ('running', 'VBG'
), ('the', 'DT'), ('``', '``'), ('Jelly', 'RB'), Tree('GPE', [('Bean', 'NNP')]), ("''", "
''"), ('and', 'CC'), ('``', '``'), ('Ice', 'NNP'), ('Cream', 'NNP'), ('Sandwich', 'NNP'),
("''", "''"), ('operating', 'VBG'), ('systems', 'NNS'), (',', ','), ('which', 'WDT'), Tre
e('PERSON', [('Apple', 'NNP')]), ('claims', 'VBZ'), ('infringe', 'VB'), ('its', 'PRP$'),
('patents', 'NNS'), ('.', '.')]), Tree('S', [('The', 'DT'), ('six', 'CD'), ('phones', 'NN
S'), ('and', 'CC'), ('tablets', 'NNS'), ('affected', 'VBN'), ('are', 'VBP'), ('the', 'DT'
), Tree('ORGANIZATION', [('Galaxy', 'NNP')]), ('S', 'NNP'), ('III', 'NNP'), (',', ','), (
'running', 'VBG'), ('the', 'DT'), ('new', 'JJ'), Tree('PERSON', [('Jelly', 'NNP'), ('Bean
', 'NNP')]), ('system', 'NN'), (',', ','), ('the', 'DT'), Tree('ORGANIZATION', [('Galaxy'
, 'NNP')]), ('Tab', 'NNP'), ('8.9', 'CD'), ('Wifi', 'NNP'), ('tablet', 'NN'), (',', ','),
('the', 'DT'), Tree('ORGANIZATION', [('Galaxy', 'NNP')]), ('Tab', 'NNP'), ('2', 'CD'), ('
10.1', 'CD'), (',', ','), Tree('PERSON', [('Galaxy', 'NNP'), ('Rugby', 'NNP'), ('Pro', 'N
NP')]), ('and', 'CC'), Tree('PERSON', [('Galaxy', 'NNP'), ('S', 'NNP')]), ('III', 'NNP'),
```

```
('mini', 'NN'), ('.', '.')]), Tree('S', [Tree('PERSON', [('Apple', 'NNP')]), ('stated', '
VBD'), ('it', 'PRP'), ('had', 'VBD'), ('^', 'NNP'), ('acted', 'VBD'), ('quickly', 'RB'),
('and', 'CC'), ('diligently', 'RB'), ("''", "''"), ('in', 'IN'), ('order', 'NN'), ('to',
'TO'), ('``', '``'), ('determine', 'VB'), ('that', 'IN'), ('these', 'DT'), ('newly', 'RB'
), ('released', 'VBN'), ('products', 'NNS'), ('do', 'VBP'), ('infringe', 'VB'), ('many',
'JJ'), ('of', 'IN'), ('the', 'DT'), ('same', 'JJ'), ('claims', 'NNS'), ('already', 'RB'),
('asserted', 'VBN'), ('by', 'IN'), Tree('PERSON', [('Apple', 'NNP')]), ('.', '.'), ("''",
"''")]), Tree('S', [('In', 'IN'), Tree('GPE', [('August', 'NNP')]), (',', ','), Tree('PER
SON', [('Samsung', 'NNP')]), ('lost', 'VBD'), ('a', 'DT'), Tree('GSP', [('US', 'NNP')]),
('patent', 'NN'), ('case', 'NN'), ('to', 'TO'), Tree('GPE', [('Apple', 'NNP')]), ('and',
'CC'), ('was', 'VBD'), ('ordered', 'VBN'), ('to', 'TO'), ('pay', 'VB'), ('its', 'PRP$'),
('rival', 'JJ'), ('$', '$'), ('1.05bn', 'CD'), ('(', '('), ('£0.66bn', 'NN'), (')', ')'),
('in', 'IN'), ('damages', 'NNS'), ('for', 'IN'), ('copying', 'VBG'), ('features', 'NNS'),
('of', 'IN'), ('the', 'DT'), Tree('ORGANIZATION', [('iPad', 'NN')]), ('and', 'CC'), Tree(
'ORGANIZATION', [('iPhone', 'NN')]), ('in', 'IN'), ('its', 'PRP$'), Tree('GPE', [('Galaxy
', 'NNP')]), ('range', 'NN'), ('of', 'IN'), ('devices', 'NNS'), ('.', '.')]), Tree('S', [
Tree('GPE', [('Samsung', 'NNP')]), (',', ','), ('which', 'WDT'), ('is', 'VBZ'), ('the', '
DT'), ('world', 'NN'), ("'s", 'POS'), ('top', 'JJ'), ('mobile', 'NN'), ('phone', 'NN'), (
'maker', 'NN'), (',', ','), ('is', 'VBZ'), ('appealing', 'VBG'), ('the', 'DT'), ('ruling'
, 'NN'), ('.', '.')]), Tree('S', [('A', 'DT'), ('similar', 'JJ'), ('case', 'NN'), ('in',
'IN'), ('the', 'DT'), Tree('ORGANIZATION', [('UK', 'NNP')]), ('found', 'VBD'), ('in', 'IN
'), Tree('GPE', [('Samsung', 'NNP')]), ("'s", 'POS'), ('favour', 'NN'), ('and', 'CC'), ('
ordered', 'VBD'), Tree('PERSON', [('Apple', 'NNP')]), ('to', 'TO'), ('publish', 'VB'), ('
an', 'DT'), ('apology', 'NN'), ('making', 'VBG'), ('clear', 'JJ'), ('that', 'IN'), ('the'
, 'DT'), Tree('LOCATION', [('South', 'JJ'), ('Korean', 'JJ')]), ('firm', 'NN'), ('had', '
VBD'), ('not', 'RB'), ('copied', 'VBN'), ('its', 'PRP$'), ('iPad', 'NN'), ('when', 'WRB')
, ('designing', 'VBG'), ('its', 'PRP$'), ('own', 'JJ'), ('devices', 'NNS'), ('.', '.')])]
```

[points: 2] Exercise 1c: Constituency parsing

Use the `nltk.RegexpParser` to perform constituency parsing on each sentence.

Use `print` to show the output in the notebook (and hence also in the exported PDF!).

In [128]:

```
constituent_parser = nltk.RegexpParser('''
NP: {<DT>? <JJ>* <NN>*} # NP
P: {<IN>}           # Preposition
V: {<V.*>}          # Verb
PP: {<P> <NP>}      # PP -> P NP
VP: {<V> <NP|PP>*}  # VP -> V (NP|PP)*''')
```

In [129]:

```
constituency_output_per_sentence = []
for tags in pos_tags_per_sentence:
    constituent_structure = constituent_parser.parse(tags)
    constituency_output_per_sentence.append(constituent_structure)
```

In [130]:

```
print(constituency_output_per_sentence)
```

```
[Tree('S', [Tree('NP', [('https', 'NN')]), (':', ':'), Tree('NP', [('//www.telegraph.co.u
k/technology/apple/9702716/Apple-Samsung-lawsuit-six-more-products-under-scrutiny.html',
'JJ')]), ('Documents', 'NNS'), Tree('VP', [Tree('V', [('filed', 'VBN')])]), ('to', 'TO'),
Tree('NP', [('the', 'DT')]), ('San', 'NNP'), ('Jose', 'NNP'), Tree('NP', [('federal', 'JJ
'), ('court', 'NN')]), Tree('P', [('in', 'IN')]), ('California', 'NNP'), Tree('P', [('on'
, 'IN')]), ('November', 'NNP'), ('23', 'CD'), Tree('NP', [('list', 'NN')]), ('six', 'CD')
, ('Samsung', 'NNP'), ('products', 'NNS'), Tree('VP', [Tree('V', [('running', 'VBG')]), T
ree('NP', [('the', 'DT')])]), ('``', '``'), ('Jelly', 'RB'), ('Bean', 'NNP'), ("''", "''"
), ('and', 'CC'), ('``', '``'), ('Ice', 'NNP'), ('Cream', 'NNP'), ('Sandwich', 'NNP'), ("
''", "''"), Tree('VP', [Tree('V', [('operating', 'VBG')])]), ('systems', 'NNS'), (',', ',
'), ('which', 'WDT'), ('Apple', 'NNP'), Tree('VP', [Tree('V', [('claims', 'VBZ')])]), Tre
e('VP', [Tree('V', [('infringe', 'VB')])]), ('its', 'PRP$'), ('patents', 'NNS'), ('.', '.
')]), Tree('S', [Tree('NP', [('The', 'DT')]), ('six', 'CD'), ('phones', 'NNS'), ('and', '
```

```
CC'), ('tablets', 'NNS'), Tree('VP', [Tree('V', [('affected', 'VBN')])]), Tree('VP', [Tre
e('V', [('are', 'VBP')]), Tree('NP', [('the', 'DT')])]), ('Galaxy', 'NNP'), ('S', 'NNP'),
('III', 'NNP'), (',', ','), Tree('VP', [Tree('V', [('running', 'VBG')]), Tree('NP', [('th
e', 'DT'), ('new', 'JJ')])]), ('Jelly', 'NNP'), ('Bean', 'NNP'), Tree('NP', [('system', '
NN')]), (',', ','), Tree('NP', [('the', 'DT')]), ('Galaxy', 'NNP'), ('Tab', 'NNP'), ('8.9
', 'CD'), ('Wifi', 'NNP'), Tree('NP', [('tablet', 'NN')]), (',', ','), Tree('NP', [('the'
, 'DT')]), ('Galaxy', 'NNP'), ('Tab', 'NNP'), ('2', 'CD'), ('10.1', 'CD'), (',', ','), ('
Galaxy', 'NNP'), ('Rugby', 'NNP'), ('Pro', 'NNP'), ('and', 'CC'), ('Galaxy', 'NNP'), ('S'
, 'NNP'), ('III', 'NNP'), Tree('NP', [('mini', 'NN')]), ('.', '.')]), Tree('S', [('Apple'
, 'NNP'), Tree('VP', [Tree('V', [('stated', 'VBD')])]), ('it', 'PRP'), Tree('VP', [Tree('
V', [('had', 'VBD')])]), ('``', 'NNP'), Tree('VP', [Tree('V', [('acted', 'VBD')])]), ('qui
ckly', 'RB'), ('and', 'CC'), ('diligently', 'RB'), ("''", "''"), Tree('PP', [Tree('P', [(
'in', 'IN')]), Tree('NP', [('order', 'NN')])]), ('to', 'TO'), ('``', '``'), Tree('VP', [T
ree('V', [('determine', 'VB')]), Tree('PP', [Tree('P', [('that', 'IN')]), Tree('NP', [('t
hese', 'DT')])])])]), ('newly', 'RB'), Tree('VP', [Tree('V', [('released', 'VBN')])]), ('pr
oducts', 'NNS'), Tree('VP', [Tree('V', [('do', 'VBP')])]), Tree('VP', [Tree('V', [('infri
nge', 'VB')]), Tree('NP', [('many', 'JJ')]), Tree('PP', [Tree('P', [('of', 'IN')]), Tree(
'NP', [('the', 'DT'), ('same', 'JJ')])])]), ('claims', 'NNS'), ('already', 'RB'), Tree('V
P', [Tree('V', [('asserted', 'VBN')])]), Tree('P', [('by', 'IN')]), ('Apple', 'NNP'), ('.
', '.'), ("''", "''")]), Tree('S', [Tree('P', [('In', 'IN')]), ('August', 'NNP'), (',', '
,'), ('Samsung', 'NNP'), Tree('VP', [Tree('V', [('lost', 'VBD')]), Tree('NP', [('a', 'DT'
)])])]), ('US', 'NNP'), Tree('NP', [('patent', 'NN'), ('case', 'NN')]), ('to', 'TO'), ('App
le', 'NNP'), ('and', 'CC'), Tree('VP', [Tree('V', [('was', 'VBD')])]), Tree('VP', [Tree('
V', [('ordered', 'VBN')])]), ('to', 'TO'), Tree('VP', [Tree('V', [('pay', 'VB')])]), ('it
s', 'PRP$'), Tree('NP', [('rival', 'JJ')]), ('$', '$'), ('1.05bn', 'CD'), ('(', '('), Tre
e('NP', [('£0.66bn', 'NN')]), (')', ')'), Tree('P', [('in', 'IN')]), ('damages', 'NNS'),
Tree('P', [('for', 'IN')]), Tree('VP', [Tree('V', [('copying', 'VBG')])]), ('features', '
NNS'), Tree('PP', [Tree('P', [('of', 'IN')]), Tree('NP', [('the', 'DT'), ('iPad', 'NN')])
]), ('and', 'CC'), Tree('NP', [('iPhone', 'NN')]), Tree('P', [('in', 'IN')]), ('its', 'PR
P$'), ('Galaxy', 'NNP'), Tree('NP', [('range', 'NN')]), Tree('P', [('of', 'IN')]), ('devi
ces', 'NNS'), ('.', '.')]), Tree('S', [('Samsung', 'NNP'), (',', ','), ('which', 'WDT'),
Tree('VP', [Tree('V', [('is', 'VBZ')]), Tree('NP', [('the', 'DT'), ('world', 'NN')])]), (
"'s", 'POS'), Tree('NP', [('top', 'JJ'), ('mobile', 'NN'), ('phone', 'NN'), ('maker', 'NN
')]), (',', ','), Tree('VP', [Tree('V', [('is', 'VBZ')])]), Tree('VP', [Tree('V', [('appe
aling', 'VBG')]), Tree('NP', [('the', 'DT'), ('ruling', 'NN')])]), ('.', '.')]), Tree('S'
, [Tree('NP', [('A', 'DT'), ('similar', 'JJ'), ('case', 'NN')]), Tree('PP', [Tree('P', [(
'in', 'IN')]), Tree('NP', [('the', 'DT')])]), ('UK', 'NNP'), Tree('VP', [Tree('V', [('fou
nd', 'VBD')])]), Tree('P', [('in', 'IN')]), ('Samsung', 'NNP'), ("'s", 'POS'), Tree('NP',
[('favour', 'NN')]), ('and', 'CC'), Tree('VP', [Tree('V', [('ordered', 'VBD')])]), ('Appl
e', 'NNP'), ('to', 'TO'), Tree('VP', [Tree('V', [('publish', 'VB')]), Tree('NP', [('an',
'DT'), ('apology', 'NN')])]), Tree('VP', [Tree('V', [('making', 'VBG')]), Tree('NP', [('c
lear', 'JJ')]), Tree('PP', [Tree('P', [('that', 'IN')]), Tree('NP', [('the', 'DT'), ('Sou
th', 'JJ'), ('Korean', 'JJ'), ('firm', 'NN')])])]), Tree('VP', [Tree('V', [('had', 'VBD')
])]), ('not', 'RB'), Tree('VP', [Tree('V', [('copied', 'VBN')])]), ('its', 'PRP$'), Tree(
'NP', [('iPad', 'NN')]), ('when', 'WRB'), Tree('VP', [Tree('V', [('designing', 'VBG')])])
, ('its', 'PRP$'), Tree('NP', [('own', 'JJ')]), ('devices', 'NNS'), ('.', '.')])]
```

In [131]:

```
for elem in constituency_output_per_sentence:
    elem.draw()

# We added some visualization of the tree structures. It may be the case that many window
s will open after
# each other as there are many sentences. We attempted to save the output of the tree str
ucture, however;
# due to the length of the output when converting the jupyter notebook to PDF, it always
cuts most of
# the information out and our TA agreed that having this bit of code is enough.
```

Augment the RegexpParser so that it also detects Named Entity Phrases (NEP), e.g., that it detects *Galaxy S III* and *Ice Cream Sandwich*

In [132]:

```
constituent_parser_v2 = nltk.RegexpParser('''
NP: {<DT>? <JJ>* <NN>*} # NP
```

```
P: {<IN>}           # Preposition
V: {<V.*>}          # Verb
PP: {<P> <NP>}      # PP -> P NP
VP: {<V> <NP|PP>*}  # VP -> V (NP|PP)*
NEP: {<JJ>* <NNP>+} # Named Entity Phrase (NEP)''')

# In the parser <JJ>* means -> Optional adjectives before an entity (e.g. 'Big Apple')
#<NNP>+ means -> 1 or more consecutive proper nouns (e.g. 'Galaxy S III', 'Ice Cream Sandwich')
```

In [133]:

```
# Testing the code above to see if it works on the provided example, we are aware that this is not part
# of the assignment, however; we would like to show that the code works as intended/described

text_ = "I love my Galaxy S III and Ice Cream Sandwich."

tokens = word_tokenize(text_)
pos_tags = nltk.pos_tag(tokens)
tree = constituent_parser_v2.parse(pos_tags)
print(tree)
```

```
(S
  I/PRP
  (VP (V love/VBP))
  my/PRP$
  (NEP Galaxy/NNP S/NNP III/NNP)
  and/CC
  (NEP Ice/NNP Cream/NNP Sandwich/NNP)
  ./.)
```

In [134]:

```
constituency_v2_output_per_sentence = []
for tags_v2 in pos_tags_per_sentence:
    constituent_structure_v2 = constituent_parser_v2.parse(tags_v2)
    constituency_v2_output_per_sentence.append(constituent_structure_v2)
```

In [135]:

```
print(constituency_v2_output_per_sentence)
```

```
[Tree('S', [Tree('NP', [('https', 'NN')]), (':', ':'), Tree('NP', [('//www.telegraph.co.uk/technology/apple/9702716/Apple-Samsung-lawsuit-six-more-products-under-scrutiny.html', 'JJ')]), ('Documents', 'NNS'), Tree('VP', [Tree('V', [('filed', 'VBN')])]), ('to', 'TO'), Tree('NP', [('the', 'DT')]), Tree('NEP', [('San', 'NNP'), ('Jose', 'NNP')]), Tree('NP', [('federal', 'JJ'), ('court', 'NN')]), Tree('P', [('in', 'IN')]), Tree('NEP', [('California', 'NNP')]), Tree('P', [('on', 'IN')]), Tree('NEP', [('November', 'NNP')]), ('23', 'CD'), Tree('NP', [('list', 'NN')]), ('six', 'CD'), Tree('NEP', [('Samsung', 'NNP')]), ('products', 'NNS'), Tree('VP', [Tree('V', [('running', 'VBG')]), Tree('NP', [('the', 'DT')])]), ('``', '``'), ('Jelly', 'RB'), Tree('NEP', [('Bean', 'NNP')]), ("''", "''"), ('and', 'CC'), ('``', '``'), Tree('NEP', [('Ice', 'NNP'), ('Cream', 'NNP'), ('Sandwich', 'NNP')]), ("''", "''"), Tree('VP', [Tree('V', [('operating', 'VBG')])]), ('systems', 'NNS'), (',', ','), ('which', 'WDT'), Tree('NEP', [('Apple', 'NNP')]), Tree('VP', [Tree('V', [('claims', 'VBZ')])]), Tree('VP', [Tree('V', [('infringe', 'VB')])]), ('its', 'PRP$'), ('patents', 'NNS'), ('.', '.')]), Tree('S', [Tree('NP', [('The', 'DT')]), ('six', 'CD'), ('phones', 'NNS'), ('and', 'CC'), ('tablets', 'NNS'), Tree('VP', [Tree('V', [('affected', 'VBN')])]), Tree('VP', [Tree('V', [('are', 'VBP')]), Tree('NP', [('the', 'DT')])]), Tree('NEP', [('Galaxy', 'NNP'), ('S', 'NNP'), ('III', 'NNP')]), (',', ','), Tree('VP', [Tree('V', [('running', 'VBG')]), Tree('NP', [('the', 'DT'), ('new', 'JJ')])]), Tree('NEP', [('Jelly', 'NNP'), ('Bean', 'NNP')]), Tree('NP', [('system', 'NN')]), (',', ','), Tree('NP', [('the', 'DT')]), Tree('NEP', [('Galaxy', 'NNP'), ('Tab', 'NNP')]), ('8.9', 'CD'), Tree('NEP', [('Wifi', 'NNP')]), Tree('NP', [('tablet', 'NN')]), (',', ','), Tree('NP', [('the', 'DT')]), Tree('NEP', [('Galaxy', 'NNP'), ('Tab', 'NNP')]), ('2', 'CD'), ('10.1', 'CD'), (',', ','), Tree('NEP', [('Galaxy', 'NNP'), ('Rugby', 'NNP'), ('Pro', 'NNP')]), ('and', 'CC'), Tree('NEP', [('Galaxy', 'NNP'), ('S', 'NNP'), ('III', 'NNP')]), Tree('NP', [('mini', 'NN')]), ('.', '.')]), Tree('S', [Tree('NEP', [('Apple', 'NNP')]), Tree('VP', [Tree('V', [('stated'
```

```
, 'VBD')])]), ('it', 'PRP'), Tree('VP', [Tree('V', [('had', 'VBD')])]), Tree('NEP', [('^'
, 'NNP')]), Tree('VP', [Tree('V', [('acted', 'VBD')])]), ('quickly', 'RB'), ('and', 'CC')
, ('diligently', 'RB'), ("'", "'"), Tree('PP', [Tree('P', [('in', 'IN')]), Tree('NP', [
('order', 'NN')])]), ('to', 'TO'), ('``', '``'), Tree('VP', [Tree('V', [('determine', 'VB
')]), Tree('PP', [Tree('P', [('that', 'IN')]), Tree('NP', [('these', 'DT')])])]), ('newly
', 'RB'), Tree('VP', [Tree('V', [('released', 'VBN')])]), ('products', 'NNS'), Tree('VP',
[Tree('V', [('do', 'VBP')])]), Tree('VP', [Tree('V', [('infringe', 'VB')]), Tree('NP', [(
'many', 'JJ')]), Tree('PP', [Tree('P', [('of', 'IN')]), Tree('NP', [('the', 'DT'), ('same
', 'JJ')])])])]), ('claims', 'NNS'), ('already', 'RB'), Tree('VP', [Tree('V', [('asserted',
'VBN')])]), Tree('P', [('by', 'IN')]), Tree('NEP', [('Apple', 'NNP')]), ('.', '.'), ("'"
, "'")]), Tree('S', [Tree('P', [('In', 'IN')]), Tree('NEP', [('August', 'NNP')]), (',',
',',), Tree('NEP', [('Samsung', 'NNP')]), Tree('VP', [Tree('V', [('lost', 'VBD')]), Tree('
NP', [('a', 'DT')])]), Tree('NEP', [('US', 'NNP')]), Tree('NP', [('patent', 'NN'), ('case
', 'NN')]), ('to', 'TO'), Tree('NEP', [('Apple', 'NNP')]), ('and', 'CC'), Tree('VP', [Tre
e('V', [('was', 'VBD')])]), Tree('VP', [Tree('V', [('ordered', 'VBN')])]), ('to', 'TO'),
Tree('VP', [Tree('V', [('pay', 'VB')])]), ('its', 'PRP$'), Tree('NP', [('rival', 'JJ')]),
('$', '$'), ('1.05bn', 'CD'), ('(', '('), Tree('NP', [('£0.66bn', 'NN')]), (')', ')'), Tr
ee('P', [('in', 'IN')]), ('damages', 'NNS'), Tree('P', [('for', 'IN')]), Tree('VP', [Tree
('V', [('copying', 'VBG')])]), ('features', 'NNS'), Tree('PP', [Tree('P', [('of', 'IN')])
, Tree('NP', [('the', 'DT'), ('iPad', 'NN')])]), ('and', 'CC'), Tree('NP', [('iPhone', 'N
N')]), Tree('P', [('in', 'IN')]), ('its', 'PRP$'), Tree('NEP', [('Galaxy', 'NNP')]), Tree
('NP', [('range', 'NN')]), Tree('P', [('of', 'IN')]), ('devices', 'NNS'), ('.', '.')]), T
ree('S', [Tree('NEP', [('Samsung', 'NNP')]), (',', ','), ('which', 'WDT'), Tree('VP', [Tr
ee('V', [('is', 'VBZ')]), Tree('NP', [('the', 'DT'), ('world', 'NN')])]), ("'s", 'POS'),
Tree('NP', [('top', 'JJ'), ('mobile', 'NN'), ('phone', 'NN'), ('maker', 'NN')]), (',', ',
'), Tree('VP', [Tree('V', [('is', 'VBZ')])]), Tree('VP', [Tree('V', [('appealing', 'VBG')
]), Tree('NP', [('the', 'DT'), ('ruling', 'NN')])]), ('.', '.')]), Tree('S', [Tree('NP',
[('A', 'DT'), ('similar', 'JJ'), ('case', 'NN')]), Tree('PP', [Tree('P', [('in', 'IN')]),
Tree('NP', [('the', 'DT')])]), Tree('NEP', [('UK', 'NNP')]), Tree('VP', [Tree('V', [('fou
nd', 'VBD')])]), Tree('P', [('in', 'IN')]), Tree('NEP', [('Samsung', 'NNP')]), ("'s", 'PO
S'), Tree('NP', [('favour', 'NN')]), ('and', 'CC'), Tree('VP', [Tree('V', [('ordered', 'V
BD')])]), Tree('NEP', [('Apple', 'NNP')]), ('to', 'TO'), Tree('VP', [Tree('V', [('publish
', 'VB')]), Tree('NP', [('an', 'DT'), ('apology', 'NN')])]), Tree('VP', [Tree('V', [('mak
ing', 'VBG')]), Tree('NP', [('clear', 'JJ')]), Tree('PP', [Tree('P', [('that', 'IN')]), T
ree('NP', [('the', 'DT'), ('South', 'JJ'), ('Korean', 'JJ'), ('firm', 'NN')])])]), Tree('
VP', [Tree('V', [('had', 'VBD')])]), ('not', 'RB'), Tree('VP', [Tree('V', [('copied', 'VB
N')])])]), ('its', 'PRP$'), Tree('NP', [('iPad', 'NN')]), ('when', 'WRB'), Tree('VP', [Tree
('V', [('designing', 'VBG')])]), ('its', 'PRP$'), Tree('NP', [('own', 'JJ')]), ('devices'
, 'NNS'), ('.', '.')])]
```

In [136]:

```
for elem_v2 in constituency_v2_output_per_sentence:
    elem_v2.draw()
```

[total points: 1] Exercise 2: spaCy

Use Spacy to process the same text as you analyzed with NLTK.

pos tagging, named entity recognition, constituency parsing

In [137]:

```
import spacy
nlp = spacy.load('en_core_web_sm')
# When loading the language model for spaCy it was not possible to do it by simply using
"en" in the
# parameter section of 'spacy.load(....)' because using simply 'en' was deprecated in thi
s version of
# spaCy and replaced with the command you see above; we were further instructed by the TA
's that it
# is not necessary to try and get the 'en' parameter to work as this does the same. We tr
ust that this
# will not have an effect on the grading of the assignment due to this situation
```

In [150]:

```
doc = nlp(text)
sentences = list(doc.sents)

# Sentence splitting and tokenization

pos_tags_spacy = []
dependency_parsing_spaCy = []

for sentence in sentences:
    for token in sentence:
        # pos tagging
        pos_tags_spacy.append((token.text, token.pos_, token.tag_))
        # dependency parsing
        dependency_parsing_spaCy.append((token.head, token.dep_, token.text))
```

In [151]:

```
print(pos_tags_spacy)
```

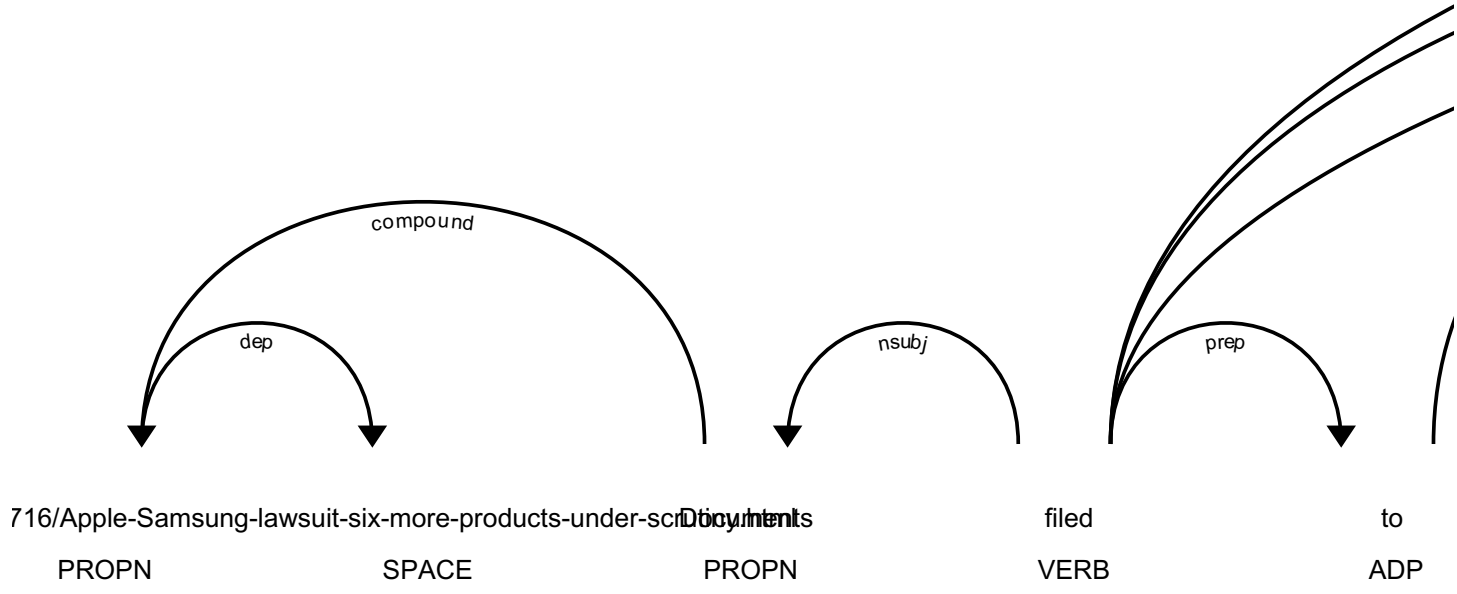
```
[('https://www.telegraph.co.uk/technology/apple/9702716/Apple-Samsung-lawsuit-six-more-pr
oducts-under-scrutiny.html', 'PROPN', 'NNP'), ('\n\n', 'SPACE', '_SP'), ('Documents', 'PR
OPN', 'NNPS'), ('filed', 'VERB', 'VBD'), ('to', 'ADP', 'IN'), ('the', 'DET', 'DT'), ('San
', 'PROPN', 'NNP'), ('Jose', 'PROPN', 'NNP'), ('federal', 'ADJ', 'JJ'), ('court', 'NOUN',
'NN'), ('in', 'ADP', 'IN'), ('California', 'PROPN', 'NNP'), ('on', 'ADP', 'IN'), ('Novemb
er', 'PROPN', 'NNP'), ('23', 'NUM', 'CD'), ('list', 'NOUN', 'NN'), ('six', 'NUM', 'CD'),
('Samsung', 'PROPN', 'NNP'), ('products', 'NOUN', 'NNS'), ('running', 'VERB', 'VBG'), ('t
he', 'DET', 'DT'), ('"', 'PUNCT', '``'), ('Jelly', 'PROPN', 'NNP'), ('Bean', 'PROPN', 'NN
P'), ('"', 'PUNCT', "''"), ('and', 'CCONJ', 'CC'), ('"', 'PUNCT', '``'), ('Ice', 'PROPN',
'NNP'), ('Cream', 'PROPN', 'NNP'), ('Sandwich', 'PROPN', 'NNP'), ('"', 'PUNCT', "''"), ('
operating', 'NOUN', 'NN'), ('systems', 'NOUN', 'NNS'), (',', 'PUNCT', ','), ('which', 'PR
ON', 'WDT'), ('Apple', 'PROPN', 'NNP'), ('claims', 'VERB', 'VBZ'), ('infringe', 'VERB', '
VBP'), ('its', 'PRON', 'PRP$'), ('patents', 'NOUN', 'NNS'), ('.', 'PUNCT', '.'), ('\n', '
SPACE', '_SP'), ('The', 'DET', 'DT'), ('six', 'NUM', 'CD'), ('phones', 'NOUN', 'NNS'), ('
and', 'CCONJ', 'CC'), ('tablets', 'NOUN', 'NNS'), ('affected', 'VERB', 'VBN'), ('are', 'A
UX', 'VBP'), ('the', 'DET', 'DT'), ('Galaxy', 'PROPN', 'NNP'), ('S', 'PROPN', 'NNP'), ('I
II', 'PROPN', 'NNP'), (',', 'PUNCT', ','), ('running', 'VERB', 'VBG'), ('the', 'DET', 'DT
'), ('new', 'ADJ', 'JJ'), ('Jelly', 'PROPN', 'NNP'), ('Bean', 'PROPN', 'NNP'), ('system',
'NOUN', 'NN'), (',', 'PUNCT', ','), ('the', 'DET', 'DT'), ('Galaxy', 'PROPN', 'NNP'), ('T
ab', 'PROPN', 'NNP'), ('8.9', 'NUM', 'CD'), ('Wifi', 'PROPN', 'NNP'), ('tablet', 'NOUN',
'NN'), (',', 'PUNCT', ','), ('the', 'DET', 'DT'), ('Galaxy', 'PROPN', 'NNP'), ('Tab', 'PR
OPN', 'NNP'), ('2', 'NUM', 'CD'), ('10.1', 'NUM', 'CD'), (',', 'PUNCT', ','), ('Galaxy',
'PROPN', 'NNP'), ('Rugby', 'PROPN', 'NNP'), ('Pro', 'PROPN', 'NNP'), ('and', 'CCONJ', 'CC
'), ('Galaxy', 'PROPN', 'NNP'), ('S', 'PROPN', 'NNP'), ('III', 'PROPN', 'NNP'), ('mini',
'NOUN', 'NN'), ('.', 'PUNCT', '.'), ('\n', 'SPACE', '_SP'), ('Apple', 'PROPN', 'NNP'), ('
stated', 'VERB', 'VBD'), ('it', 'PRON', 'PRP'), ('had', 'AUX', 'VBD'), ('"', 'PUNCT', '``
'), ('acted', 'VERB', 'VBN'), ('quickly', 'ADV', 'RB'), ('and', 'CCONJ', 'CC'), ('diligen
tly', 'ADV', 'RB'), ('"', 'PUNCT', "''"), ('in', 'ADP', 'IN'), ('order', 'NOUN', 'NN'), (
'to', 'PART', 'TO'), ('"', 'PUNCT', '``'), ('determine', 'VERB', 'VB'), ('that', 'SCONJ',
'IN'), ('these', 'DET', 'DT'), ('newly', 'ADV', 'RB'), ('released', 'VERB', 'VBN'), ('pro
ducts', 'NOUN', 'NNS'), ('do', 'AUX', 'VBP'), ('infringe', 'VERB', 'VB'), ('many', 'ADJ',
'JJ'), ('of', 'ADP', 'IN'), ('the', 'DET', 'DT'), ('same', 'ADJ', 'JJ'), ('claims', 'NOUN
', 'NNS'), ('already', 'ADV', 'RB'), ('asserted', 'VERB', 'VBN'), ('by', 'ADP', 'IN'), ('
Apple', 'PROPN', 'NNP'), ('.', 'PUNCT', '.'), ('"', 'PUNCT', "''"), ('\n', 'SPACE', '_SP'
), ('In', 'ADP', 'IN'), ('August', 'PROPN', 'NNP'), (',', 'PUNCT', ','), ('Samsung', 'PRO
PN', 'NNP'), ('lost', 'VERB', 'VBD'), ('a', 'DET', 'DT'), ('US', 'PROPN', 'NNP'), ('paten
t', 'NOUN', 'NN'), ('case', 'NOUN', 'NN'), ('to', 'ADP', 'IN'), ('Apple', 'PROPN', 'NNP')
, ('and', 'CCONJ', 'CC'), ('was', 'AUX', 'VBD'), ('ordered', 'VERB', 'VBN'), ('to', 'PART
', 'TO'), ('pay', 'VERB', 'VB'), ('its', 'PRON', 'PRP$'), ('rival', 'ADJ', 'JJ'), ('$', '
SYM', '$'), ('1.05bn', 'NUM', 'CD'), ('(', 'PUNCT', '-LRB-'), ('£', 'SYM', '$'), ('0.66bn
', 'NUM', 'CD'), (')', 'PUNCT', '-RRB-'), ('in', 'ADP', 'IN'), ('damages', 'NOUN', 'NNS')
, ('for', 'ADP', 'IN'), ('copying', 'VERB', 'VBG'), ('features', 'NOUN', 'NNS'), ('of', '
ADP', 'IN'), ('the', 'DET', 'DT'), ('iPad', 'PROPN', 'NNP'), ('and', 'CCONJ', 'CC'), ('iP
hone', 'PROPN', 'NNP'), ('in', 'ADP', 'IN'), ('its', 'PRON', 'PRP$'), ('Galaxy', 'PROPN',
'NNP'), ('range', 'NOUN', 'NN'), ('of', 'ADP', 'IN'), ('devices', 'NOUN', 'NNS'), ('.', '
PUNCT', '.'), ('Samsung', 'PROPN', 'NNP'), (',', 'PUNCT', ','), ('which', 'PRON', 'WDT'),
('is', 'AUX', 'VBZ'), ('the', 'DET', 'DT'), ('world', 'NOUN', 'NN'), ("'s", 'PART', 'POS'
), ('top', 'ADJ', 'JJ'), ('mobile', 'ADJ', 'JJ'), ('phone', 'NOUN', 'NN'), ('maker', 'NOU
N', 'NN'), (',', 'PUNCT', ','), ('is', 'AUX', 'VBZ'), ('appealing', 'VERB', 'VBG'), ('the
```

```
', 'DET', 'DT'), ('ruling', 'NOUN', 'NN'), ('.', 'PUNCT', '.'), ('\n', 'SPACE', '_SP'), (
'A', 'DET', 'DT'), ('similar', 'ADJ', 'JJ'), ('case', 'NOUN', 'NN'), ('in', 'ADP', 'IN'),
('the', 'DET', 'DT'), ('UK', 'PROPN', 'NNP'), ('found', 'VERB', 'VBD'), ('in', 'ADP', 'IN
'), ('Samsung', 'PROPN', 'NNP'), ("'s", 'PART', 'POS'), ('favour', 'NOUN', 'NN'), ('and',
'CCONJ', 'CC'), ('ordered', 'VERB', 'VBD'), ('Apple', 'PROPN', 'NNP'), ('to', 'PART', 'TO
'), ('publish', 'VERB', 'VB'), ('an', 'DET', 'DT'), ('apology', 'NOUN', 'NN'), ('making',
'NOUN', 'NN'), ('clear', 'ADJ', 'JJ'), ('that', 'SCONJ', 'IN'), ('the', 'DET', 'DT'), ('S
outh', 'ADJ', 'JJ'), ('Korean', 'ADJ', 'JJ'), ('firm', 'NOUN', 'NN'), ('had', 'AUX', 'VBD
'), ('not', 'PART', 'RB'), ('copied', 'VERB', 'VBN'), ('its', 'PRON', 'PRP$'), ('iPad', '
PROPN', 'NNP'), ('when', 'SCONJ', 'WRB'), ('designing', 'VERB', 'VBG'), ('its', 'PRON', '
PRP$'), ('own', 'ADJ', 'JJ'), ('devices', 'NOUN', 'NNS'), ('.', 'PUNCT', '.')]
```

In [153]:

```
from spacy import displacy
displacy.render(doc, jupyter=True, style='dep')

# We added this because we felt that it shows the full output of the NER and the dependen
cy parsing
```



In [154]:

```
print(dependency_parsing_spaCy)

# In this code cell is the output of the list where each tuple is (starting word, relatio
n between starting
# and next word, next word) although this might seem excessive. However, it is clearer to
see the relation
# in this format when compared to the output of the previous code cell
```

```
[(Documents, 'compound', 'https://www.telegraph.co.uk/technology/apple/9702716/Apple-Sams
ung-lawsuit-six-more-products-under-scrutiny.html'), (https://www.telegraph.co.uk/technol
ogy/apple/9702716/Apple-Samsung-lawsuit-six-more-products-under-scrutiny.html, 'dep', '\n
\n'), (filed, 'nsubj', 'Documents'), (filed, 'ROOT', 'filed'), (filed, 'prep', 'to'), (co
urt, 'det', 'the'), (Jose, 'compound', 'San'), (court, 'nmod', 'Jose'), (court, 'amod', '
federal'), (to, 'pobj', 'court'), (court, 'prep', 'in'), (in, 'pobj', 'California'), (fil
ed, 'prep', 'on'), (on, 'pobj', 'November'), (November, 'nummod', '23'), (filed, 'npadvmo
d', 'list'), (products, 'nummod', 'six'), (products, 'compound', 'Samsung'), (list, 'appo
s', 'products'), (products, 'acl', 'running'), (Bean, 'det', 'the'), (Bean, 'punct', '"')
, (Bean, 'compound', 'Jelly'), (running, 'dobj', 'Bean'), (Bean, 'punct', '"'), (Bean, 'c
c', 'and'), (Sandwich, 'punct', '"'), (Cream, 'compound', 'Ice'), (Sandwich, 'compound',
'Cream'), (systems, 'nmod', 'Sandwich'), (Sandwich, 'punct', '"'), (systems, 'compound',
'operating'), (Bean, 'conj', 'systems'), (systems, 'punct', ','), (claims, 'nsubj', 'whic
h'), (claims, 'nsubj', 'Apple'), (systems, 'relcl', 'claims'), (claims, 'xcomp', 'infring
e'), (patents, 'poss', 'its'), (infringe, 'dobj', 'patents'), (filed, 'punct', '.'), (.,
'dep', '\n'), (phones, 'det', 'The'), (phones, 'nummod', 'six'), (are, 'nsubj', 'phones')
, (phones, 'cc', 'and'), (phones, 'conj', 'tablets'), (phones, 'acl', 'affected'), (are,
'ROOT', 'are'), (III, 'det', 'the'), (III, 'compound', 'Galaxy'), (III, 'compound', 'S'),
(are, 'attr', 'III'), (are, 'punct', ','), (are, 'advcl', 'running'), (system, 'det', 'th
e'), (system, 'amod', 'new'), (Bean, 'compound', 'Jelly'), (system, 'compound', 'Bean'),
(running, 'dobj', 'system'), (running, 'punct', ','), (tablet, 'det', 'the'), (Tab, 'comp
ound', 'Galaxy'), (tablet, 'nmod', 'Tab'), (Tab, 'nummod', '8.9'), (tablet, 'compound', '
Wifi'), (are, 'dep', 'tablet'), (tablet, 'punct', ','), (Tab, 'det', 'the'), (Tab, 'compo
und', 'Galaxy'), (tablet, 'appos', 'Tab'), (10.1, 'nummod', '2'), (Tab, 'appos', '10.1'),
(tablet, 'punct', ','), (Pro, 'compound', 'Galaxy'), (Pro, 'compound', 'Rugby'), (tablet,
'appos', 'Pro'), (Pro, 'cc', 'and'), (S, 'compound', 'Galaxy'), (III, 'compound', 'S'), (
Pro, 'conj', 'III'), (Pro, 'conj', 'mini'), (are, 'punct', '.'), (., 'dep', '\n'), (state
d, 'nsubj', 'Apple'), (stated, 'ROOT', 'stated'), (acted, 'nsubj', 'it'), (acted, 'aux',
'had'), (acted, 'punct', '^'), (stated, 'ccomp', 'acted'), (acted, 'advmod', 'quickly'),
(quickly, 'cc', 'and'), (quickly, 'conj', 'diligently'), (acted, 'punct', '"'), (acted, '
prep', 'in'), (in, 'pobj', 'order'), (determine, 'aux', 'to'), (determine, 'punct', '"'),
(order, 'acl', 'determine'), (infringe, 'mark', 'that'), (products, 'det', 'these'), (rel
eased, 'advmod', 'newly'), (products, 'amod', 'released'), (infringe, 'nsubj', 'products'
), (infringe, 'aux', 'do'), (determine, 'ccomp', 'infringe'), (infringe, 'dobj', 'many'),
(many, 'prep', 'of'), (claims, 'det', 'the'), (claims, 'amod', 'same'), (of, 'pobj', 'cla
ims'), (asserted, 'advmod', 'already'), (claims, 'acl', 'asserted'), (asserted, 'agent',
'by'), (by, 'pobj', 'Apple'), (stated, 'punct', '.'), (lost, 'punct', '"'), (", 'dep', '\
n'), (lost, 'prep', 'In'), (In, 'pobj', 'August'), (lost, 'punct', ','), (lost, 'nsubj',
'Samsung'), (lost, 'ROOT', 'lost'), (case, 'det', 'a'), (case, 'compound', 'US'), (case,
'compound', 'patent'), (lost, 'dobj', 'case'), (lost, 'prep', 'to'), (to, 'pobj', 'Apple'
), (lost, 'cc', 'and'), (ordered, 'auxpass', 'was'), (lost, 'conj', 'ordered'), (pay, 'au
x', 'to'), (ordered, 'xcomp', 'pay'), (rival, 'poss', 'its'), (pay, 'dobj', 'rival'), (1.
05bn, 'nmod', '$'), (rival, 'appos', '1.05bn'), (1.05bn, 'punct', '('), (0.66bn, 'nmod',
'£'), (1.05bn, 'appos', '0.66bn'), (1.05bn, 'punct', ')'), (pay, 'prep', 'in'), (in, 'pob
j', 'damages'), (damages, 'prep', 'for'), (for, 'pcomp', 'copying'), (copying, 'dobj', 'f
eatures'), (features, 'prep', 'of'), (iPad, 'det', 'the'), (of, 'pobj', 'iPad'), (iPad, '
cc', 'and'), (iPad, 'conj', 'iPhone'), (copying, 'prep', 'in'), (range, 'poss', 'its'), (
range, 'compound', 'Galaxy'), (in, 'pobj', 'range'), (range, 'prep', 'of'), (of, 'pobj',
'devices'), (lost, 'punct', '.'), (appealing, 'nsubj', 'Samsung'), (Samsung, 'punct', ','
), (is, 'nsubj', 'which'), (Samsung, 'relcl', 'is'), (world, 'det', 'the'), (maker, 'poss
', 'world'), (world, 'case', "'s"), (maker, 'amod', 'top'), (phone, 'amod', 'mobile'), (m
aker, 'compound', 'phone'), (is, 'attr', 'maker'), (Samsung, 'punct', ','), (appealing, '
aux', 'is'), (appealing, 'ROOT', 'appealing'), (ruling, 'det', 'the'), (appealing, 'dobj'
, 'ruling'), (appealing, 'punct', '.'), (., 'dep', '\n'), (case, 'det', 'A'), (case, 'amo
d', 'similar'), (case, 'ROOT', 'case'), (case, 'prep', 'in'), (UK, 'det', 'the'), (in, 'p
obj', 'UK'), (case, 'acl', 'found'), (found, 'prep', 'in'), (favour, 'poss', 'Samsung'),
(Samsung, 'case', "'s"), (in, 'pobj', 'favour'), (found, 'cc', 'and'), (found, 'conj', 'o
rdered'), (ordered, 'dobj', 'Apple'), (publish, 'aux', 'to'), (ordered, 'xcomp', 'publish
'), (making, 'det', 'an'), (making, 'compound', 'apology'), (publish, 'dobj', 'making'),
(making, 'amod', 'clear'), (copied, 'mark', 'that'), (firm, 'det', 'the'), (Korean, 'amod
', 'South'), (firm, 'amod', 'Korean'), (copied, 'nsubj', 'firm'), (copied, 'aux', 'had'),
(copied, 'neg', 'not'), (ordered, 'ccomp', 'copied'), (iPad, 'poss', 'its'), (copied, 'do
```

```
(copied, neg, not), (ordered, ccomp, copied), (iPad, poss, its), (copied, do
bj', 'iPad'), (designing, 'advmod', 'when'), (copied, 'advcl', 'designing'), (devices, 'p
oss', 'its'), (devices, 'amod', 'own'), (designing, 'dobj', 'devices'), (case, 'punct', '
.')]
```

In [142]:

```
from spacy import displacy

tree_structure = displacy.render(doc, jupyter=False, style='dep')

output_path = "spaCy_tree_Lab1.svg"
with open(output_path, "w") as text_file:
    text_file.write(tree_structure)
```

In [144]:

```
# named entity recognition

entities_spaCy = []

displacy.render(doc, jupyter=True, style='ent')

print("----------------------------------------------------------------------------------
-----------------")

for ent in doc.ents:
    entities_spaCy.append((ent.text, ent.label_))

print(entities_spaCy)
```

https://www.telegraph.co.uk/technology/apple/9702716/Apple-Samsung-lawsuit-six-more-products-under-scrutiny.html

Documents filed to the San Jose GPE federal court in California GPE on November 23 DATE list six CARDINAL Samsung ORG products running the " Jelly Bean WORK_OF_ART " and "Ice Cream Sandwich" operating systems, which Apple ORG claims infringe its patents.

The six CARDINAL phones and tablets affected are the Galaxy S III PERSON , running the new Jelly Bean ORG system, the Galaxy Tab 8.9 Wifi PERSON tablet, the Galaxy Tab 2 10.1 PERSON , Galaxy Rugby Pro and Galaxy S III mini.

Apple ORG stated it had “acted quickly and diligently" in order to "determine that these newly released products do infringe many of the same claims already asserted by Apple ORG ."

In August DATE , Samsung ORG lost a US GPE patent case to Apple ORG and was ordered to pay its rival $ 1.05bn MONEY (£ 0.66bn MONEY) in damages for copying features of the iPad ORG and iPhone ORG in its Galaxy range of devices. Samsung ORG , which is the world's top mobile phone maker, is appealing the ruling.

A similar case in the UK GPE found in Samsung ORG 's favour and ordered Apple ORG to publish an apology making clear that the South Korean NORP firm had not copied its iPad ORG when designing its own devices.

```
--------------------------------------------------------------------------------
----------
[('San Jose', 'GPE'), ('California', 'GPE'), ('November 23', 'DATE'), ('six', 'CARDINAL')
, ('Samsung', 'ORG'), ('Jelly Bean', 'WORK_OF_ART'), ('Apple', 'ORG'), ('six', 'CARDINAL'
), ('the Galaxy S III', 'PERSON'), ('Jelly Bean', 'ORG'), ('Galaxy Tab 8.9 Wifi', 'PERSON
'), ('Galaxy Tab 2 10.1', 'PERSON'), ('Apple', 'ORG'), ('Apple', 'ORG'), ('August', 'DATE
'), ('Samsung', 'ORG'), ('US', 'GPE'), ('Apple', 'ORG'), ('1.05bn', 'MONEY'), ('0.66bn',
'MONEY'), ('iPad', 'ORG'), ('iPhone', 'ORG'), ('Samsung', 'ORG'), ('UK', 'GPE'), ('Samsun
g', 'ORG'), ('Apple', 'ORG'), ('South Korean', 'NORP'), ('iPad', 'ORG')]
```

small tip: You can use `sents = list(doc.sents)` to be able to use the index to access a sentence like `sents[2]` for the third sentence.

[total points: 7] Exercise 3: Comparison NLTK and spaCy

We will now compare the output of NLTK and spaCy, i.e., in what do they differ?

[points: 3] Exercise 3a: Part of speech tagging

Compare the output from NLTK and spaCy regarding part of speech tagging.

- **To compare, you probably would like to compare sentence per sentence. Describe if the sentence splitting is different for NLTK than for spaCy. If not, where do they differ?**

After looking at the output of both NLTK and spaCy, we can see some interesting differences in the way that the different libraries split the text. For NLTK, being an older library and keeping the more traditional rule-based approaches and Machine Learning models, means that it is not as comprehensive or easy to use out of the box; some more work has to be done than in this assignment to enhance its performance. That being said, spaCy is not perfect either but does seem to outperform nltk for sentence splitting.

Here are some differences that were noticed:

- **For spaCy, the link was treated as a single token when being split which is the correct way to split an url; however, for the NLTK splitter the link was split into 3 different tokens which technically is incorrect since an url must be preserved in order to keep its meaning (since an url is a single entity, fragmenting it can destroy its context) and these days it is standard NLP practice to keep an url as a single token for consistency.**
- **Another interesting element which we noticed is that spaCy seems to also give spaces their own tokens whereas NLTK does not. Although this seems trivial at first, it is good practice to give spaces their own tokens to preserve whitespace information which allows for more precise text processing. This can be seen as an advantage since it can lead to better-formatting analysis due to preserving the total sentence structure, thus also allowing for more formatting-sensitive NLP tasks to be achieved which rely on such structures being preserved (e.g. medical domain, legal domain, financial domain).**

After checking the sentence splitting, select a sentence for which you expect interesting results and perhaps differences. Motivate your choice.

- **We choose to check the second sentence for further analysis due to the heavy use of varying language, numerical text, and complex grammar which we believe will result in different pos tagging from the different libraries. We make this hypothesis because spaCy is built on more modern, statistical Machine Learning frameworks compared to NLTK. In addition, spaCy is built for more streamline commercial use which means that it is more complete and more user-friendly to use, whereas NLTK is older, needs more code, and also requires more manual work to accomplish the same task.**

Compare the output in `token.tag` from spaCy to the part of speech tagging from NLTK for each token in your selected sentence. Are there any differences? This is not a trick question; it is possible that there are no differences.

For the selected sentence (sentence 2) there are some interesting differences:

The → NLTK(DT), spaCy(DET, DT) → both correctly tagged this as a determiner

six → NLTK(CD), spaCy(NUM, CD) → both correctly tagged this as a number

phones → NLTK(NNS), spaCy(NOUN, NNS) → both correctly tagged as noun

and → NLTK(CC), spaCy(CCONJ, CC) → both correctly tagged as coordinating conjunction

tablets → NLTK(NNS), spaCy (NOUN, NNS) → both correctly tagged

affected → NLTK(VBN), spaCy(VERB, VBN) → both incorrectly tagged as verb, here 'affected' is used as an adjective form of the verb 'affect'

are
→ NLTK(VBP), spaCy(AUX, VBP) -> NLTK incorrectly tagged as verb, non-3rd person singular present whereas spaCy correctly tagged as auxiliary verb since it is a linking verb connecting the subject to a complement

the → NLTK(DT), spaCy(DET, DT) → both correctly tagged

Galaxy → NLTK(NNP), spaCy(PROPN, NNP) → both correctly tagged, the noun of a phone is its brand name/specific model name

S → NLTK(NNP), spaCy(PROPN, NNP) → both correctly tagged, the noun of a phone is its brand name/specific model name

III → NLTK(NNP), spaCy(PROPN, NNP) → both correctly tagged, the noun of a phone is its brand name/specific model name

, → NLTK(,), spaCy(PUNCT, ',') → both correctly tagged

running
→ NLTK(VBG), spaCy(VERB, VBG)
→ both incorrectly tagged, rather than 'running' being a verb, gerund or present participle (action verb) in this context it is used as an adjective to describe that the mobiles are operating on a specific piece of OS software

the → NLTK(DT), spaCy(DET, DT) → both correctly tagged

new → NLTK(JJ), spaCy(ADJ, JJ) → both correctly tagged, describes the state/version of the OS software system

Jelly → NLTK(NNP), spaCy(PROPN, NNP) → both correctly tagged

Bean → NLTK(NNP), spaCy(PROPN, NNP) → both correctly tagged

system → NLTK(NN), spaCy(NOUN, NN) → both correctly tagged, noun singular or mass

, → NLTK(,), spaCy(PUNCT, ',') → both correctly tagged

the → NLTK(DT), spaCy(DET, DT) → both correctly tagged

Galaxy → NLTK(NNP), spaCy(PROPN, NNP) → both correctly tagged, the noun of a phone is its brand name/specific model name

Tab → NLTK(NNP), spaCy(PROPN, NNP) → both correctly tagged, the noun of a phone is its brand name/specific model name

8.9
→ NLTK(CD), spaCy(NUM, CD)
→ both incorrectly tagged, although it is a cardinal number and would be correct if this was a standalone number, in this context this number is still a part of the name for the device (noun) and therefore should still be tagged as NNP, otherwise there are inconsistencies

Wifi → NLTK(NNP), spaCy(PROPN, NNP) → both correctly tagged

tablet → NLTK(NN), spaCy(NOUN, NN) → both correctly tagged, noun singular or mass

, → NLTK(,), spaCy(PUNCT, ',') → both correctly tagged

the → NLTK(DT), spaCy(DET, DT) → both correctly tagged

Galaxy → NLTK(NNP), spaCy(PROPN, NNP) → both correctly tagged, the noun of a phone is its brand name/specific model name

Tab → NLTK(NNP), spaCy(PROPN, NNP) -> both correctly tagged, the noun of a phone is its brand name/specific model name

2 NLTK(CD), spaCy(NUM, CD)
→ both incorrectly tagged, although it is a cardinal number and would be correct if this was a standalone number, in this context this number is still a part of the name for the device (noun) and therefore should still be tagged as NNP, otherwise there are inconsistencies

10.1
→ NLTK(CD), spaCy(NUM, CD)
→ both incorrectly tagged, although it is a cardinal number and would be correct if this was a standalone number, in this context this number is still a part of the name for the device (noun) and therefore should still be tagged as NNP, otherwise there are inconsistencies

, → NLTK(,), spaCy(PUNCT, ',') → both correctly tagged

Galaxy → NLTK(NNP), spaCy(PROPN, NNP) → both correctly tagged

Rugby → NLTK(NNP), spaCy(PROPN, NNP) → both correctly tagged

Pro → NLTK(NNP), spaCy(PROPN, NNP) → both correctly tagged

and → NLTK(CC), spaCy(CCONJ, CC) → both correctly tagged

Galaxy → NLTK(NNP), spaCy(PROPN, NNP) → both correctly tagged

S → NLTK(NNP), spaCy(PROPN, NNP) → both correctly tagged

III → NLTK(NNP), spaCy(PROPN, NNP) → both correctly tagged

mini
→ NLTK(NN), spaCy(NOUN, NN)
→ both incorrectly tagged, 'mini' is not a standalone noun here, it instead modifies the name of the device (proper noun) where 'mini' indicates a smaller version of the device name, therefore it is technically not correct to tag it as a standalone noun

. → NLTK(.), spaCy(PUNCT, ',') → both correctly tagged

\n
→ NLTK(not detected by NLTK), spaCy(SPACE, _SP)
→ This is debatable as mentioned in the previous answer, however; for completeness and commercial pipelines it is more useful to retain the structure of a sentence as much as possible

Above are the differences between the libraries, and the incorrectly tagged words are marked in red whereas the different tags are marked in orange.

Although there are not as many differences as we originally thought (due to the simplicity of the text we had to process), the difference was when spaCy tagged 'are' as an auxiliary verb instead of non-3rd person singular, tagging the token as an auxiliary verb is more suitable based on the context of the sentence and shows some slightly better performance over NLTK. Lastly, the only other difference as has been previously mentioned is the complete sentence structure that spaCy retains (e.g. all punctuation receives a token) whereas NLTK discards these tokens as it is not considered as important. Overall, we would like to stress that due to the simplicity of the text being analyzed, and the chosen sentence, there cannot be a multitude of differences. To see more differences, a more complex text/sentence would need to be analyzed; on the other hand, we already see some interesting differences in how both of these libraries tokenize a text and what is considered important in each.

[points: 2] Exercise 3b: Named Entity Recognition (NER)

- **Describe differences between the output from NLTK and spaCy for Named Entity Recognition. Which one do you think performs better?**

First sentence:

- **NLTK -> 'San Jose federal court in' (ORGANIZATION) | spaCy -> 'San Jose' (GPE)**
- **NLTK -> 'California on November 23, list six' (GPE) | spaCy -> 'California' (GPE), 'November 23' (DATE), 'six' (CARDINAL)**
- **NLTK -> 'Samsung product running the "Jelly' (GPE) | spaCy -> 'Samsung' (ORGANIZATION), 'Jelly Bean' (WORK_OF_ART)**
- **NLTK -> 'Bean", and "Ice Cream Sandwich" operating systems, which' (GPE) | spaCy -> ...**
- **NLTK -> 'Apple claims infringe its patents.' (PERSON) | spaCy -> 'Apple' (ORGANIZATION)**

We will not include the other sentences to keep the text brief, however; having looked closely at the output of both libraries, they both still make very silly mistakes such as saying that 'Apple' is a person because of the verb 'claim', NLTK assumes that such a verb is possessive to a human and thus makes an underlying assumption. Looking at spaCy, it also makes silly mistakes such as recognizing 'Jelly Bean' as a work of art instead of something closer to an operating system. Overall, spaCy seems to outperform NLTK by a good margin; we say this because NLTK groups many parts of a sentence together and assigns these sections 1 entity, whereas; spaCy does not do this grouping and can give an entity to each part of the text which makes it more effective in finding entities. However, as stated, both systems are not highly effective at recognizing entities "out of the box," we believe that more work must be done to enhance the performance.

[points: 2] Exercise 3c: Constituency/dependency parsing

Choose one sentence from the text and run constituency parsing using NLTK and dependency parsing using spaCy.

- **describe briefly the difference between constituency parsing and dependency parsing**

NLTK can analyze various phrases if the rules are added to the regexParser, therefore; it is more dynamic and can be used to identify more combinations of sentence structures than spaCy, however; the rules have to be made by a human and there is no relationship between the entities, simply a grouping of entities based on the rules. On the other hand, spaCy's output does not have these groupings but makes connections between the entities within the sentence. It depends on the application of what an engineer is doing whether either is more useful than the other; the NLTK regexParser may be useful because there can be an unlimited number of rules defined and found, but there is a lack of context linking. If dependencies are important for understanding contextual elements for sentiment analysis or hate speech detection, additionally; using context clues as seen in spaCy may be more useful to understand how the beginning of a sentence may relate to the middle or end.

The spaCy library does not need any human input to define the rules in the regexParser, and while that makes spaCy less dynamic than NLTK, it does mean that it will most likely generate a different output than the human rule-based approach (NLTK), making it more user-friendly and robust, meaning it is better-suited for commercial usage.

- **describe differences between the output from NLTK and spaCy.**

<u>Analyzing sentence 1</u>

An example of the different outputs:

- **NLTK -> (NEP): 'San Jose'**
- **spaCy -> (Jose, compound, San), (court, nmod, Jose)**

Analysis of differing outputs:

From this sentence we can see that there is a lot of grammatical and syntactical information extracted from the NLTK output, the regexParser is extremely useful in identifying various grammatical structures within the

sentence. This gives a nice insight into the hierarchical phrase structures present and is useful when doing full syntactic analysis (grammar checking). The output from spaCy identifies grammatical relationships between words using directed dependencies, which is useful for information extraction and as mentioned above, understanding word relationships in context. Overall, the NLTK constituency parser is more powerful in academic and research setting, whereas spaCy is more powerful in commercial products.

End of this notebook