Practical No: 1

Practical 1 a) **Date: 28/02/2023**

Aim: Install NLTK

Code:

pip install nltk

Output:

```
In [1]: pip install nltk

Requirement already satisfied: nltk in c:\users\lenovo\appdata\local\programs\python\python310\lib\site-packages (3.8.1)
Requirement already satisfied: joblib in c:\users\lenovo\appdata\local\programs\python\python310\lib\site-packages (from nltk)
(1.2.0)
Requirement already satisfied: click in c:\users\lenovo\appdata\local\programs\python\python310\lib\site-packages (from nltk)
(8.1.3)
Requirement already satisfied: regex>=2021.8.3 in c:\users\lenovo\appdata\local\programs\python\python310\lib\site-packages (fr
om nltk) (2022.10.31)
Requirement already satisfied: tqdm in c:\users\lenovo\appdata\local\programs\python\python310\lib\site-packages (from nltk)
(4.64.1)
Requirement already satisfied: colorama in c:\users\lenovo\appdata\local\programs\python\python310\lib\site-packages (from clic
k->nltk) (0.4.6)
Note: you may need to restart the kernel to use updated packages.


[notice] A new release of pip is available: 23.0.1 -> 23.1.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```


Roll No: Name:

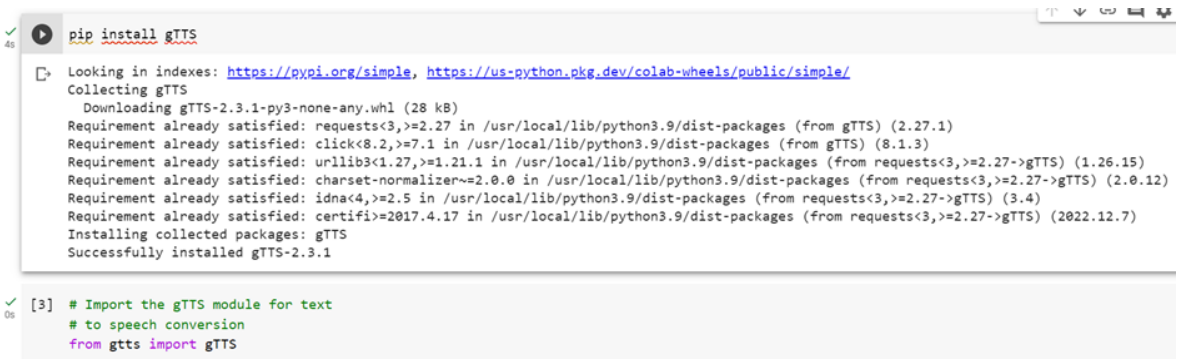
Practical 1 b) **Date: 28/02/2023**

Aim: Convert the given text to speech.

Code:

```
pip install gTTS
# Import the gTTS module for text
# to speech conversion
from gtts import gTTS
pip install playsound
# This module is imported so that we can
# play the converted audio
from playsound import playsound
# It is a text value that we want to convert to audio
text_val = 'All the best for your exam.'
# Here are converting in English Language
language = 'en'
# Passing the text and language to the engine,
# here we have assign slow=False. Which denotes
# the module that the transformed audio should
# have a high speed
obj = gTTS(text=text_val, lang=language, slow=False)
#Here we are saving the transformed audio in a mp3 file named
# exam.mp3
obj.save("exam.mp3")
# Play the exam.mp3 file
playsound("exam.mp3")
```

Output:



Roll No: Name:

```
[5] pip install playsound

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting playsound
  Downloading playsound-1.3.0.tar.gz (7.7 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: playsound
  Building wheel for playsound (setup.py) ... done
  Created wheel for playsound: filename=playsound-1.3.0-py3-none-any.whl size=7035 sha256=9854d89287e12dfd582667381ed8dad94cc17f5c6649d
  Stored in directory: /root/.cache/pip/wheels/ba/39/54/c8f7ff9a88a644d3c58b4dec802d90b79a2e0fb2a6b884bf82
Successfully built playsound
Installing collected packages: playsound
Successfully installed playsound-1.3.0
```

```
[6] # This module is imported so that we can
    # play the converted audio

    from playsound import playsound

WARNING:playsound:playsound is relying on another python subprocess. Please use `pip install pygobject` if you want playsound to run mor
```

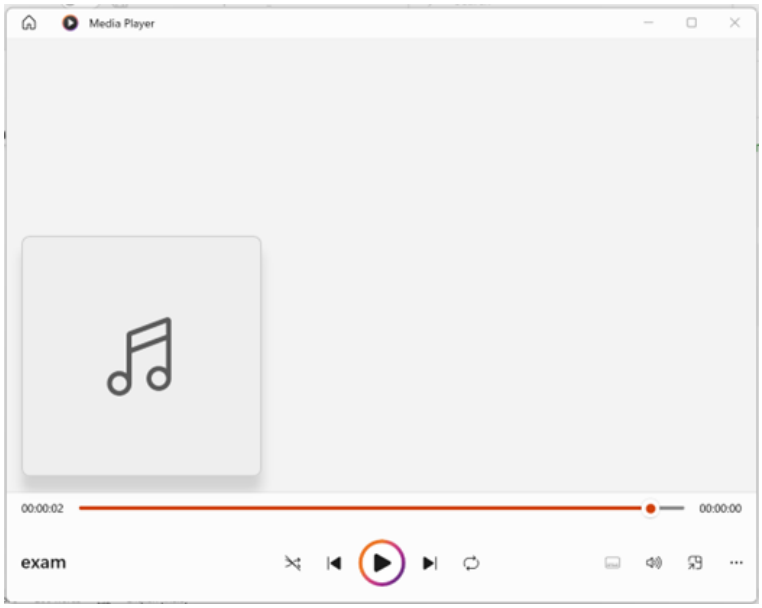
```
[7] # It is a text value that we want to convert to audio
    text_val = 'All the best for your exam.'
```

```
    # Here are converting in English Language
    language = 'en'
```

```
[9] # Passing the text and language to the engine,
    # here we have assign slow=False. Which denotes
    # the module that the transformed audio should
    # have a high speed
    obj = gTTS(text=text_val, lang=language, slow=False)
```

```
[10] #Here we are saving the transformed audio in a mp3 file named
     # exam.mp3
     obj.save("exam.mp3")
```

```
[11]   # Play the exam.mp3 file
     playsound("exam.mp3")
```



Roll No: Name:

Practical 1 c) **Date: 28/02/2023**

Aim: Convert the Speech of .wav audio file to Text.

Code:

```
!pip install SpeechRecognition pydub

#import library

import speech_recognition as sr

r= sr.Recognizer()

with sr.AudioFile("male.wav")as source:

audio_data = r.record(source)

text = r.recognize_google(audio_data)

print(text)
```

Output:

Roll No: Name:

Practical No: 2

Practical 2 a) **Date: 14-03-23**

Aim: Study of various Corpus – Brown, Inaugural, Reuters, udhr with various methods like filelds, raw, words, sents, categories.

Code:

```
import nltk
from nltk.corpus import brown
nltk.download('brown')
print('File ids of brown corpus\n',brown.fileids())
ca01 = brown.words('ca01')
print('\nca01 has following words:\n',ca01)
print('\nca01 has', len(ca01),'words')
print('\n\nCategories or file in brown corpus:\n')
print(brown.categories())
print('\n\nStatistics for each text:\n')
print('AvgWordLen\tAvgSentenceLen\tno.ofTimesEachWordAppearsOnAvg\t\tFileName')
for fileid in brown.fileids():
    num_chars = len(brown.raw(fileid))
    num_words = len(brown.words(fileid))
    num_sents = len(brown.sents(fileid))
    num_vocab = len(set([w.lower() for w in brown.words(fileid)]))
    print(int(num_chars/num_words),'\t\t\t', int(num_words/num_sents),'\t\t\t',
int(num_words/num_vocab),'\t\t\t',fileid)
```

Output:

```
In [1]: import nltk

In [2]: from nltk.corpus import brown

In [3]: nltk.download('brown')
        [nltk_data] Downloading package brown to
        [nltk_data]     C:\Users\admin\AppData\Roaming\nltk_data...
        [nltk_data]   Unzipping corpora\brown.zip.
Out[3]: True
```


Roll No: Name:

```
print('File ids of brown corpus\n',brown.fileids())
```

```
File ids of brown corpus
 ['ca01', 'ca02', 'ca03', 'ca04', 'ca05', 'ca06', 'ca07', 'ca08', 'ca09', 'ca10', 'ca11', 'ca12', 'ca13', 'ca14', 'ca15', 'ca1
6', 'ca17', 'ca18', 'ca19', 'ca20', 'ca21', 'ca22', 'ca23', 'ca24', 'ca25', 'ca26', 'ca27', 'ca28', 'ca29', 'ca30', 'ca31', 'ca
32', 'ca33', 'ca34', 'ca35', 'ca36', 'ca37', 'ca38', 'ca39', 'ca40', 'ca41', 'ca42', 'ca43', 'ca44', 'cb01', 'cb02', 'cb03', 'c
b04', 'cb05', 'cb06', 'cb07', 'cb08', 'cb09', 'cb10', 'cb11', 'cb12', 'cb13', 'cb14', 'cb15', 'cb16', 'cb17', 'cb18', 'cb19',
'cb20', 'cb21', 'cb22', 'cb23', 'cb24', 'cb25', 'cb26', 'cb27', 'cc01', 'cc02', 'cc03', 'cc04', 'cc05', 'cc06', 'cc07', 'cc08',
'cc09', 'cc10', 'cc11', 'cc12', 'cc13', 'cc14', 'cc15', 'cc16', 'cc17', 'cd01', 'cd02', 'cd03', 'cd04', 'cd05', 'cd06', 'cd07',
'cd08', 'cd09', 'cd10', 'cd11', 'cd12', 'cd13', 'cd14', 'cd15', 'cd16', 'cd17', 'ce01', 'ce02', 'ce03', 'ce04', 'ce05', 'ce06',
'ce07', 'ce08', 'ce09', 'ce10', 'ce11', 'ce12', 'ce13', 'ce14', 'ce15', 'ce16', 'ce17', 'ce18', 'ce19', 'ce20', 'ce21', 'ce22',
'ce23', 'ce24', 'ce25', 'ce26', 'ce27', 'ce28', 'ce29', 'ce30', 'ce31', 'ce32', 'ce33', 'ce34', 'ce35', 'ce36', 'cf01', 'cf02',
'cf03', 'cf04', 'cf05', 'cf06', 'cf07', 'cf08', 'cf09', 'cf10', 'cf11', 'cf12', 'cf13', 'cf14', 'cf15', 'cf16', 'cf17', 'cf18',
'cf19', 'cf20', 'cf21', 'cf22', 'cf23', 'cf24', 'cf25', 'cf26', 'cf27', 'cf28', 'cf29', 'cf30', 'cf31', 'cf32', 'cf33', 'cf34',
'cf35', 'cf36', 'cf37', 'cf38', 'cf39', 'cf40', 'cf41', 'cf42', 'cf43', 'cf44', 'cf45', 'cf46', 'cf47', 'cf48', 'cg01', 'cg02',
'cg03', 'cg04', 'cg05', 'cg06', 'cg07', 'cg08', 'cg09', 'cg10', 'cg11', 'cg12', 'cg13', 'cg14', 'cg15', 'cg16', 'cg17', 'cg18',
'cg19', 'cg20', 'cg21', 'cg22', 'cg23', 'cg24', 'cg25', 'cg26', 'cg27', 'cg28', 'cg29', 'cg30', 'cg31', 'cg32', 'cg33', 'cg34',
'cg35', 'cg36', 'cg37', 'cg38', 'cg39', 'cg40', 'cg41', 'cg42', 'cg43', 'cg44', 'cg45', 'cg46', 'cg47', 'cg48', 'cg49', 'cg50',
'cg51', 'cg52', 'cg53', 'cg54', 'cg55', 'cg56', 'cg57', 'cg58', 'cg59', 'cg60', 'cg61', 'cg62', 'cg63', 'cg64', 'cg65', 'cg66',
'cg67', 'cg68', 'cg69', 'cg70', 'cg71', 'cg72', 'cg73', 'cg74', 'cg75', 'ch01', 'ch02', 'ch03', 'ch04', 'ch05', 'ch06', 'ch07',
'ch08', 'ch09', 'ch10', 'ch11', 'ch12', 'ch13', 'ch14', 'ch15', 'ch16', 'ch17', 'ch18', 'ch19', 'ch20', 'ch21', 'ch22', 'ch23',
'ch24', 'ch25', 'ch26', 'ch27', 'ch28', 'ch29', 'ch30', 'cj01', 'cj02', 'cj03', 'cj04', 'cj05', 'cj06', 'cj07', 'cj08', 'cj09',
'cj10', 'cj11', 'cj12', 'cj13', 'cj14', 'cj15', 'cj16', 'cj17', 'cj18', 'cj19', 'cj20', 'cj21', 'cj22', 'cj23', 'cj24', 'cj25',
'cj26', 'cj27', 'cj28', 'cj29', 'cj30', 'cj31', 'cj32', 'cj33', 'cj34', 'cj35', 'cj36', 'cj37', 'cj38', 'cj39', 'cj40', 'cj41'
```

```
print(brown.categories())
```

```
ca01 has following words:
 ['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]

ca01 has 2242 words


Categories or file in brown corpus:

['adventure', 'belles_lettres', 'editorial', 'fiction', 'government', 'hobbies', 'humor', 'learned', 'lore', 'mystery', 'news',
'religion', 'reviews', 'romance', 'science_fiction']
```

```
    print(int(num_chars/num_words),'\t\t\t', int(num_words/num_sents),'\t\t\t', int(num_words/num_vocab),'\t\t\t',fil
```

```
Statistics for each text:

AvgWordLen      AvgSentenceLen  no.ofTimesEachWordAppearsOnAvg          FileName
9                       22                      2                       ca01
8                       23                      2                       ca02
8                       20                      2                       ca03
9                       25                      2                       ca04
8                       26                      3                       ca05
8                       22                      2                       ca06
9                       18                      2                       ca07
8                       21                      2                       ca08
9                       19                      2                       ca09
8                       21                      2                       ca10
8                       22                      2                       ca11
8                       22                      2                       ca12
8                       20                      2                       ca13
8                       17                      2                       ca14
8                       21                      2                       ca15
8                       20                      2                       ca16
8                       22                      2                       ca17
8                       22                      2                       ca18
8                       20                      2                       ca19
8                       23                      2                       ca20
9                       20                      2                       ca21
```


Roll No: Name:

Practical 2 b) **Date: 14-03-23**

Aim: Create and use your own corpora (plaintext, categorical)

Code:

```
import nltk
from nltk.corpus import PlaintextCorpusReader
corpus_root="C:/Users/admin/Desktop/corpus"
filelist=PlaintextCorpusReader(corpus_root,'.*')
print('\n File list:\n')
print(filelist.fileids())
print(filelist.root)

import nltk
nltk.download('punkt')

print('\n\n Statistics for each text:\n')
print('AvgWordLen\tAvgSentenceLen\tno.ofTimesEachWordAppearsOnAvg\tFileName')
for fileid in filelist.fileids():
    num_chars=len(filelist.raw(fileid))
    num_words=len(filelist.words(fileid))
    num_sents=len(filelist.sents(fileid))
    num_vocab=len(set([w.lower() for w in filelist.words(fileid)]))

print(int(num_chars/num_words),'\t\t\t',int(num_words/num_sents),'\t\t\t',int(num_words/num
_vocab),'\t\t\t',fileid)
```

Output:

Roll No: Name:

```
import nltk
```

```
from nltk.corpus import PlaintextCorpusReader
```

```
corpus_root="C:/Users/lenovo/OneDrive/Desktop/corpus"
```

```
filelist=PlaintextCorpusReader(corpus_root,'.*')

print('\n File list:\n')

print(filelist.fileids())

print(filelist.root)
```

```
 File list:

['hi.txt', 'kasar.txt', 'shraddha.txt']
C:\Users\lenovo\OneDrive\Desktop\corpus
```

```
print('\n\n Statistics for each text:\n')

print('AvgWordLen\tAvgSentenceLen\tno.ofTimesEachWordAppearsOnAvg\tFileName')

for fileid in filelist.fileids():

    num_chars=len(filelist.raw(fileid))

    num_words=len(filelist.words(fileid))

    num_sents=len(filelist.sents(fileid))

    num_vocab=len(set([w.lower() for w in filelist.words(fileid)]))

    print(int(num_chars/num_words),'\t\t\t',int(num_words/num_sents),'\t\t\t',int(num_words/num_voca
```

```
 Statistics for each text:

AvgWordLen      AvgSentenceLen  no.ofTimesEachWordAppearsOnAvg  FileName
4                       1                       1                       hi.txt
7                       1                       1                       kasar.txt
10                      1                       1                       shraddha.txt
```


Roll No: Name:

Practical 2 c) **Date: 14-03-23**

Aim: Study of tagged corpora with methods like tagged_sents, tagged_words.

Code:

```
import nltk
from nltk import tokenize
nltk.download('punkt')
nltk.download('words')
para = "Hello ! My name is Shraddha Kasar . Today I'll be learning NLTK"
sents=tokenize.sent_tokenize(para)
print("\nsentence tokenization\n==========\n",sents)
print("\nword tokenization\n==========\n")
for index in range (len(sents)):
    words=tokenize.word_tokenize(sents[index])
    print(words)
```

Output:

```
: import nltk
  from nltk import tokenize
```

```
: nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\lenovo\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
: True
```

```
: nltk.download('words')
```

```
[nltk_data] Downloading package words to
[nltk_data]     C:\Users\lenovo\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\words.zip.
```

```
: True
```


Roll No: Name:

```
para = "Hello ! My name is Shraddha Kasar . Today I'll be learning NLTK"
sents=tokenize.sent_tokenize(para)
print("\nsentence tokenization\n======================\n",sents)
```

```
sentence tokenization
======================
 ['Hello !', 'My name is Shraddha Kasar .', "Today I'll be learning NLTK"]
```

```
print("\nword tokenization\n==================\n")
for index in range (len(sents)):
    words=tokenize.word_tokenize(sents[index])
    print(words)
```

```
word tokenization
==================

['Hello', '!']
['My', 'name', 'is', 'Shraddha', 'Kasar', '.']
['Today', 'I', "'ll", 'be', 'learning', 'NLTK']
```


Roll No: Name:

Practical 2 d) Date: 14-03-23

Aim: Write a program to find the most frequent noun tags.

Code:

```
import nltk
from collections import defaultdict
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
text = nltk.word_tokenize("Nick likes to play football. Nick does not like to play cricket.")
tagged = nltk.pos_tag(text)
print(tagged)
addNounWords = []
count = 0
for words in tagged:
    val = tagged[count][1]
    if(val == 'NN' or val =='NNS' or val == 'NNPS' or val == 'NNP'):
        addNounWords.append(tagged[count][0])
    count+=1
print(addNounWords)
temp = defaultdict(int)
for sub in addNounWords:
    temp[sub] += 1
res = max(temp, key = temp.get)
print("Word with maximum frequency : " + str(res))
```

Output:

Roll No: Name:

--

```
import nltk
```

```
from collections import defaultdict
```

```
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\lenovo\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
True
```

```
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\lenovo\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping taggers\averaged_perceptron_tagger.zip.
```

```
True
```

```
text = nltk.word_tokenize("Nick likes to play football. Nick does not like to play cricket.")
tagged = nltk.pos_tag(text)
print(tagged)
```

```
[('Nick', 'NNP'), ('likes', 'VBZ'), ('to', 'TO'), ('play', 'VB'), ('football', 'NN'), ('.', '.'), ('Nick', 'NNP'), ('does', 'VB
Z'), ('not', 'RB'), ('like', 'VB'), ('to', 'TO'), ('play', 'VB'), ('cricket', 'NN'), ('.', '.')]
```

```
addNounWords = []
count = 0
for words in tagged:
    val = tagged[count][1]
    if(val == 'NN' or val =='NNS' or val == 'NNPS' or val == 'NNP'):
        addNounWords.append(tagged[count][0])
    count+=1
print(addNounWords)
```

```
['Nick', 'football', 'Nick', 'cricket']
```

```
temp = defaultdict(int)
for sub in addNounWords:
    temp[sub] += 1
```

```
res = max(temp, key = temp.get)

print("Word with maximum frequency : " + str(res))
```

```
Word with maximum frequency : Nick
```


Roll No: Name:

Practical No: 3

Practical 3 a) **Date: 21/03/2023**

Aim: Study of Wordnet Dictionary with methods as synsets, definitions, examples, antonyms

Code:

```
import nltk
from nltk.corpus import wordnet
nltk.download('wordnet')
print(wordnet.synsets("computer"))
print(wordnet.synset("computer.n.01").definition())
print("Examples:", wordnet.synset("computer.n.01").examples())
print(wordnet.lemma('buy.v.01.buy').antonyms())
```

Output:

```
import nltk
from nltk.corpus import wordnet
nltk.download('wordnet')
print(wordnet.synsets("computer"))
```

```
[Synset('computer.n.01'), Synset('calculator.n.01')]
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\lenovo\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

```
print(wordnet.synset("computer.n.01").definition())
```

```
a machine for performing calculations automatically
```

```
print("Examples:", wordnet.synset("computer.n.01").examples())
```

```
Examples: []
```

```
print(wordnet.lemma('buy.v.01.buy').antonyms())
```

```
[Lemma('sell.v.01.sell')]
```


Roll No: Name:

Practical 3 b) **Date: 21/03/2023**

Aim: Study lemmas, hyponyms, hypernyms.

Code:

```
import nltk
from nltk.corpus import wordnet
nltk.download('wordnet')
print(wordnet.synsets("computer"))
print(wordnet.synset("computer.n.01").lemma_names())
for e in wordnet.synsets("computer"):
    print(f'{e} --> {e.lemma_names()}')
print(wordnet.synset('computer.n.01').lemmas())
print(wordnet.lemma('computer.n.01.computing_device').synset())
print(wordnet.lemma('computer.n.01.computing_device').name())
syn = wordnet.synset('computer.n.01')
print(syn.hyponyms)
print([lemma.name() for synset in syn.hyponyms() for lemma in synset.lemmas()])
vehicle = wordnet.synset('vehicle.n.01')
car = wordnet.synset('car.n.01')
print(car.lowest_common_hypernyms(vehicle))
```

Output:

```
import nltk
from nltk.corpus import wordnet
nltk.download('wordnet')
print(wordnet.synsets("computer"))
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\lenovo\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

```
[Synset('computer.n.01'), Synset('calculator.n.01')]
```

```
print(wordnet.synset("computer.n.01").lemma_names())
```

```
['computer', 'computing_machine', 'computing_device', 'data_processor', 'electronic_computer', 'information_processing_system']
```


Roll No: Name:

```
for e in wordnet.synsets("computer"):
    print(f'{e} --> {e.lemma_names()}')
```

```
Synset('computer.n.01') --> ['computer', 'computing_machine', 'computing_device', 'data_processor', 'electronic_computer', 'inf
ormation_processing_system']
Synset('calculator.n.01') --> ['calculator', 'reckoner', 'figurer', 'estimator', 'computer']
```

```
print(wordnet.synset('computer.n.01').lemmas())
```

```
[Lemma('computer.n.01.computer'), Lemma('computer.n.01.computing_machine'), Lemma('computer.n.01.computing_device'), Lemma('com
puter.n.01.data_processor'), Lemma('computer.n.01.electronic_computer'), Lemma('computer.n.01.information_processing_system')]
```

```
print(wordnet.lemma('computer.n.01.computing_device').synset())
```

```
Synset('computer.n.01')
```

```
print(wordnet.lemma('computer.n.01.computing_device').name())
```

```
computing_device
```

```
syn = wordnet.synset('computer.n.01')
print(syn.hyponyms)
print([lemma.name() for synset in syn.hyponyms() for lemma in synset.lemmas()])
```

```
<bound method _WordNetObject.hyponyms of Synset('computer.n.01')>
['analog_computer', 'analogue_computer', 'digital_computer', 'home_computer', 'node', 'client', 'guest', 'number_cruncher', 'pa
ri-mutuel_machine', 'totalizer', 'totaliser', 'totalizator', 'totalisator', 'predictor', 'server', 'host', 'Turing_machine', 'w
eb_site', 'website', 'internet_site', 'site']
```

```
vehicle = wordnet.synset('vehicle.n.01')
car = wordnet.synset('car.n.01')
print(car.lowest_common_hypernyms(vehicle))
```

```
[Synset('vehicle.n.01')]
```


Roll No: Name:

Practical 3 c) **Date: 21/03/2023**

Aim: sentence tokenization, word tokenization, Part of speech Tagging and chunking of user defined text.

Code:

```
import nltk
from nltk import tokenize
nltk.download('punkt')
from nltk import tag
from nltk import chunk
nltk.download('averaged_perceptron_tagger')
nltk.download('maxent_ne_chunker')
nltk.download('words')
para = "Hello! My name is Shraddha kasar. Today you'll be learning NLTK."
sents = tokenize.sent_tokenize(para)
print("\nsentence tokenization\n==================\n",sents)
print("\nword tokenization\n==================\n")
for index in range(len(sents)):para = "Hello! My name is Shraddha kasar. Today you'll be learning NLTK."
sents = tokenize.sent_tokenize(para)
print("\nsentence tokenization\n==================\n",sents)
    words = tokenize.word_tokenize(sents[index])
    print(words)
tagged_words = []
for index in range(len(sents)):
    tagged_words.append(tag.pos_tag(words))
    print("\nPOS Tagging\n=======\n")
tree = []
for index in range(len(sents)):
    tree.append(chunk.ne_chunk(tagged_words[index]))
    print("\nchunking\n=======\n")
    print(tree)
```


Roll No: Name:

Output:

```
import nltk
```

```
from nltk import tokenize
nltk.download('punkt')
from nltk import tag
from nltk import chunk
nltk.download('averaged_perceptron_tagger')
nltk.download('maxent_ne_chunker')
nltk.download('words')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\lenovo\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\lenovo\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     C:\Users\lenovo\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping chunkers\maxent_ne_chunker.zip.
[nltk_data] Downloading package words to
[nltk_data]     C:\Users\lenovo\AppData\Roaming\nltk_data...
[nltk_data]   Package words is already up-to-date!
```

```
True
```

```
para = "Hello! My name is Shraddha kasar. Today you'll be learning NLTK."
sents = tokenize.sent_tokenize(para)
print("\nsentence tokenization\n==================\n",sents)
```

```
sentence tokenization
==================
 ['Hello!', 'My name is Shraddha kasar.', "Today you'll be learning NLTK."]
```

```
print("\nword tokenization\n==================\n")
for index in range(len(sents)):para = "Hello! My name is Shraddha kasar. Today you'll be learning NLTK."
sents = tokenize.sent_tokenize(para)
print("\nsentence tokenization\n==================\n",sents)
    words = tokenize.word_tokenize(sents[index])
    print(words)
```

```
word tokenization
==================

['Hello', '!']
['My', 'name', 'is', 'Shraddha', 'kasar', '.']
['Today', 'you', "'ll", 'be', 'learning', 'NLTK', '.']
```


Roll No: Name:

```
tagged_words = []
for index in range(len(sents)):
    tagged_words.append(tag.pos_tag(words))
    print("\nPOS Tagging\n=======\n")
```

```
POS Tagging
=======


POS Tagging
=======


POS Tagging
=======
```

```
tree = []
for index in range(len(sents)):
    tree.append(chunk.ne_chunk(tagged_words[index]))
    print("\nchunking\n=======\n")
    print(tree)
```

```
chunking
=======

[Tree('S', [('Today', 'NN'), ('you', 'PRP'), ("'ll", 'MD'), ('be', 'VB'), ('learning', 'VBG'), Tree('ORGANIZATION', [('NLTK',
'NNP')]), ('.', '.')])]

chunking
=======

[Tree('S', [('Today', 'NN'), ('you', 'PRP'), ("'ll", 'MD'), ('be', 'VB'), ('learning', 'VBG'), Tree('ORGANIZATION', [('NLTK',
'NNP')]), ('.', '.')]), Tree('S', [('Today', 'NN'), ('you', 'PRP'), ("'ll", 'MD'), ('be', 'VB'), ('learning', 'VBG'), Tree('ORG
ANIZATION', [('NLTK', 'NNP')]), ('.', '.')])]

chunking
=======

[Tree('S', [('Today', 'NN'), ('you', 'PRP'), ("'ll", 'MD'), ('be', 'VB'), ('learning', 'VBG'), Tree('ORGANIZATION', [('NLTK',
'NNP')]), ('.', '.')]), Tree('S', [('Today', 'NN'), ('you', 'PRP'), ("'ll", 'MD'), ('be', 'VB'), ('learning', 'VBG'), Tree('ORG
ANIZATION', [('NLTK', 'NNP')]), ('.', '.')]), Tree('S', [('Today', 'NN'), ('you', 'PRP'), ("'ll", 'MD'), ('be', 'VB'), ('learni
ng', 'VBG'), Tree('ORGANIZATION', [('NLTK', 'NNP')]), ('.', '.')])]
```


Roll No: Name:

Practical No: 4

Practical 4 a) **Date: 05/04/2023**

Aim: Named Entity recognition using user defined text.

Code:

```
!pip install -U spacy
!python -m spacy download en_core_web_sm
import spacy
nlp = spacy.load("en_core_web_sm")
text = ("'When Sebastian Thrun started working on self-driving cars at"
    "Google in 2007, few people outside of the company took him"
    "seriously. " I can tell you senior CEOs of major American "
    "car companies would shake my hand and turn away because I wasn't"
    "worth talking to," said Thrun, in an interviwe with Recode earlier"
    "this week.'")
doc = nlp(text)
print("Noun phrases:", [chunk.text for chunk in doc.noun_chunks])
print("Verbs:", [token.lemma_ for token in doc if token.pos_ == "VERB"])
for entity in doc.ents:
  print(entity.text, entity.label_)
```

Output:

Roll No: Name:

```
print("Noun phrases:", [chunk.text for chunk in doc.noun_chunks])
print("Verbs:", [token.lemma_ for token in doc if token.pos_ == "VERB"])
```

```
Noun phrases: ['Sebastian Thrun', 'self-driving cars', '"Google', 'few people', 'the company', 'him', 'I', 'you', 'senior CEO
s', 'car companies', 'my hand', 'I', 'Thrun', 'an interviwe', 'Recode']
Verbs: ['start', 'work', 'drive', 'take', 'tell', 'shake', 'turn', 'talk', 'say']
```

```
for entity in doc.ents:
    print(entity.text, entity.label_)
```

```
Sebastian Thrun PERSON
2007 DATE
American NORP
Thrun PERSON
Recode ORG
this week DATE
```


Roll No: Name:

Practical 4 b) **Date: 05/04/2023**

Aim: Named Entity recognition with diagram using NLTK corpus – treebank.

Code:

```
import nltk

nltk.download('treebank')

from nltk.corpus import treebank_chunk

treebank_chunk.tagged_sents()[0]

treebank_chunk.chunked_sents()[0]

treebank_chunk.chunked_sents()[0].draw()
```

Output:

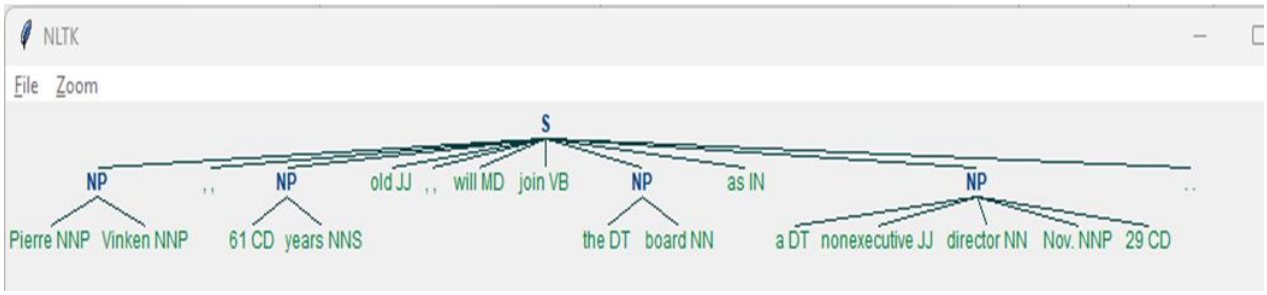
```
In [1]: import nltk

In [2]: nltk.download('treebank')
[nltk_data] Downloading package treebank to
[nltk_data]     C:\Users\admin\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\treebank.zip.
Out[2]: True

In [3]: from nltk.corpus import treebank_chunk

In [4]: treebank_chunk.tagged_sents()[0]
Out[4]: [('Pierre', 'NNP'),
         ('Vinken', 'NNP'),
         (',', ','),
         ('61', 'CD'),
         ('years', 'NNS'),
         ('old', 'JJ'),
         (',', ','),
         ('will', 'MD'),
         ('join', 'VB'),
         ('the', 'DT'),
         ('board', 'NN'),
         ('as', 'IN'),
         ('a', 'DT'),
         ('nonexecutive', 'JJ'),
         ('director', 'NN'),
         ('Nov.', 'NNP'),
         ('29', 'CD'),
         ('.', '.')]
```


Roll No: Name:

```
In [5]: treebank_chunk.chunked_sents()[0]
```

```
Out[5]: Tree('S', [Tree('NP', [('Pierre', 'NNP'), ('Vinken', 'NNP')]), (',', ','), Tree('NP', [('61', 'CD'), ('years', 'NNS')]), ('old', 'JJ'), (',', ','), ('will', 'MD'), ('join', 'VB'), Tree('NP', [('the', 'DT'), ('board', 'NN')]), ('as', 'IN'), Tree('NP', [('a', 'DT'), ('nonexecutive', 'JJ'), ('director', 'NN'), ('Nov.', 'NNP'), ('29', 'CD')]), ('.', '.')])
```

```
In [*]: treebank_chunk.chunked_sents()[0].draw()
```



Roll No: Name:

Practical No: 5

Practical 5 a) **Date: 05/04/2023**

Aim: Chart parsing using the string "Book that flight".

Code:

```
import nltk
from nltk import tokenize
grammar1 = nltk.CFG.fromstring("""
s -> VP
VP -> VP NP
NP -> Det NP
Det -> 'that'
NP -> singular Noun
NP -> 'flight'
VP -> 'Book'
""")
sentence = "Book that flight"
for index in range(len(sentence)):
    all_tokens = tokenize.word_tokenize(sentence)
print(all_tokens)
parser = nltk.ChartParser(grammar1)
for tree in parser.parse(all_tokens):
    print(tree)
    tree.draw()
```

Output:

Roll No: Name:

```
: import nltk
```

```
: from nltk import tokenize
```

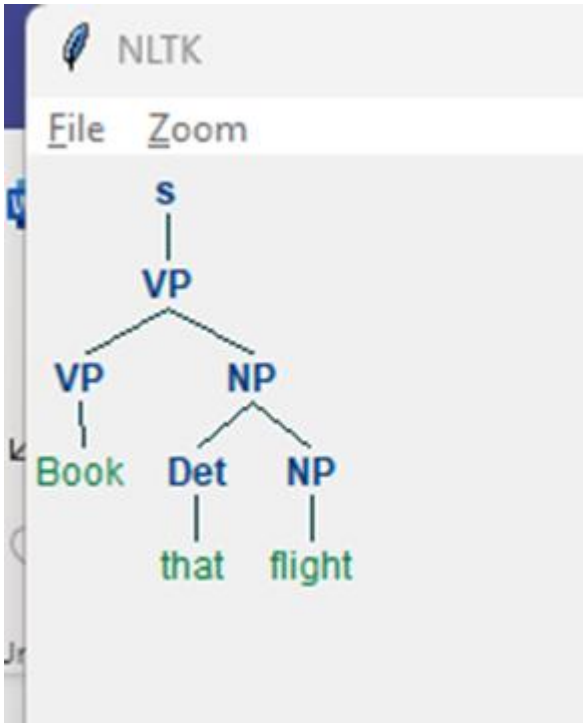
```
: grammar1 = nltk.CFG.fromstring("""
s -> VP
VP -> VP NP
NP -> Det NP
Det -> 'that'
NP -> singular Noun
NP -> 'flight'
VP -> 'Book'
""")
```

```
: sentence = "Book that flight"
```

```
: for index in range(len(sentence)):
    all_tokens = tokenize.word_tokenize(sentence)
print(all_tokens)
```

```
['Book', 'that', 'flight']
```

```
: parser = nltk.ChartParser(grammar1)
for tree in parser.parse(all_tokens):
    print(tree)
    tree.draw()
```



Roll No: Name:

Practical 5 b) **Date: 05/04/2023**

Aim: Chart parsing using the string "I saw a bird in my balcony".

Code:

```
import nltk
from nltk import tokenize
grammar1 = nltk.CFG.fromstring("""
s -> NP VP
NP -> 'I'
VP -> VP PP
VP -> V NP
V -> 'saw'
NP -> Det N
Det -> 'a'
N -> singular Noun
N -> 'bird'
PP -> P NP
P -> 'in'
NP -> Det N
Det -> 'my'
N -> 'balcony'
""")
sentence = "I saw a bird in my balcony"
for index in range(len(sentence)):
    all_tokens = tokenize.word_tokenize(sentence)
print(all_tokens)
parser = nltk.ChartParser(grammar1)
for tree in parser.parse(all_tokens):
```


Roll No: Name:

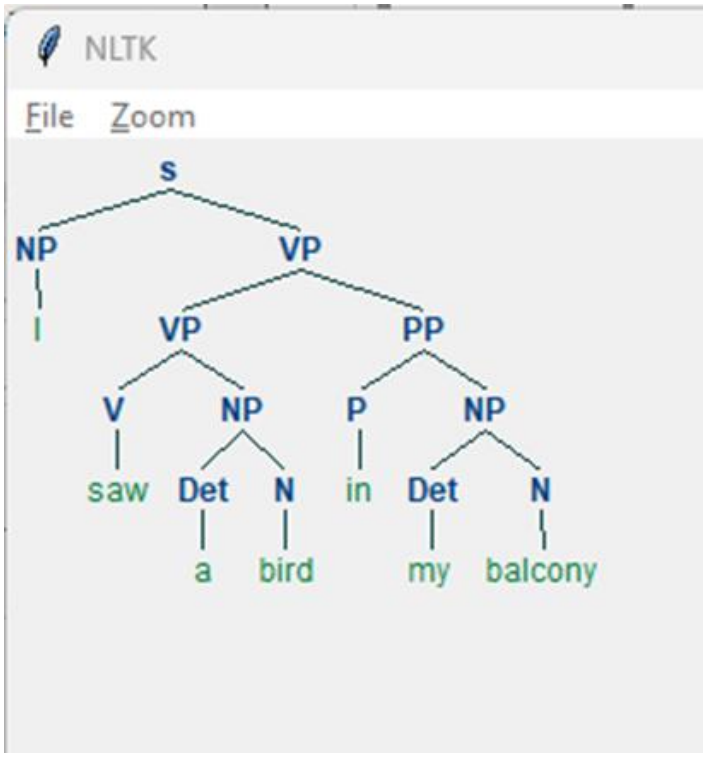
print(tree)

tree.draw()

Output:

```
In [1]: import nltk

In [2]: from nltk import tokenize

In [4]: grammar1 = nltk.CFG.fromstring("""
        s -> NP VP
        NP -> 'I'
        VP -> VP PP
        VP -> V NP
        V -> 'saw'
        NP -> Det N
        Det -> 'a'
        N -> singular Noun
        N -> 'bird'
        PP -> P NP
        P -> 'in'
        NP -> Det N
        Det -> 'my'
        N -> 'balcony'
        """)

In [5]: sentence = "I saw a bird in my balcony"

In [6]: for index in range(len(sentence)):
            all_tokens = tokenize.word_tokenize(sentence)
        print(all_tokens)

In [*]: parser = nltk.ChartParser(grammar1)
        for tree in parser.parse(all_tokens):
            print(tree)
            tree.draw()

        (s
          (NP I)
          (VP
            (VP (V saw) (NP (Det a) (N bird)))
            (PP (P in) (NP (Det my) (N balcony)))))
```


Roll No: Name:


NLTK
File Zoom
S
NP
VP
I
VP
PP
V
NP
P
NP
saw
Det
N
in
Det
N
a
bird
my
balcony

Roll No: Name:

Practical No: 6

Practical 6 a) **Date: 12/04/2023**

Aim: Analyzing the meaning of sentence by querying a database. Find the cities of India by applying a query - 'What cities are located in India' and the context free grammar from the file 'sqlIndia.fcfg'.

Hint: sqlIndia.fcfg file should be in the same folder.

Data file: sqlIndia.fcfg

```
## Natural Language Toolkit: sqlIndia.fcfg
##
## Deliberately naive string-based grammar for
## deriving SQL queries from English
##
% start S
S[SEM=(?np + WHERE + ?vp)] -> NP[SEM=?np] VP[SEM=?vp]

VP[SEM=(?v + ?pp)] -> IV[SEM=?v] PP[SEM=?pp]
VP[SEM=(?v + ?ap)] -> IV[SEM=?v] AP[SEM=?ap]
NP[SEM=(?det + ?n)] -> Det[SEM=?det] N[SEM=?n]
PP[SEM=(?p + ?np)] -> P[SEM=?p] NP[SEM=?np]
AP[SEM=?pp] -> A[SEM=?a] PP[SEM=?pp]
NP[SEM='Country="india"'] -> 'India'
Det[SEM='SELECT'] -> 'Which' | 'What'
N[SEM='City FROM city_table'] -> 'cities'
IV[SEM=''] -> 'are'
A[SEM=''] -> 'located'
P[SEM=''] -> 'in'
```

Code:

```
import nltk
from nltk import load_parser
nltk.download('book_grammars')
nltk.data.show_cfg('grammars/book_grammars/sql0.fcfg')
from nltk.parse import load_parser
cp = load_parser('grammars/book_grammars/sql0.fcfg')
query = 'What cities are located in China'
trees = list(cp.parse(query.split()))
print (trees)
answer = trees[0].label()['SEM']
print (answer)
answer = [s for s in answer if s]
q = ' '.join(answer)
print(q)
from nltk.sem import chat80
nltk.download('city_database')
rows = chat80.sql_query('corpora/city_database/city.db', q)
for r in rows:
  print(r)
```

Output:

```
import nltk
from nltk import load_parser
nltk.download('book_grammars')
```

```
[nltk_data] Downloading package book_grammars to
[nltk_data]     C:\Users\lenovo\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping grammars\book_grammars.zip.
```

```
True
```


Roll No: Name:

```
nltk.data.show_cfg('grammars/book_grammars/sql0.fcfg')
```

```
% start S
S[SEM=(?np + WHERE + ?vp)] -> NP[SEM=?np] VP[SEM=?vp]
VP[SEM=(?v + ?pp)] -> IV[SEM=?v] PP[SEM=?pp]
VP[SEM=(?v + ?ap)] -> IV[SEM=?v] AP[SEM=?ap]
NP[SEM=(?det + ?n)] -> Det[SEM=?det] N[SEM=?n]
PP[SEM=(?p + ?np)] -> P[SEM=?p] NP[SEM=?np]
AP[SEM=?pp] -> A[SEM=?a] PP[SEM=?pp]
NP[SEM='Country="greece"'] -> 'Greece'
NP[SEM='Country="china"'] -> 'China'
Det[SEM='SELECT'] -> 'Which' | 'What'
N[SEM='City FROM city_table'] -> 'cities'
IV[SEM=''] -> 'are'
A[SEM=''] -> 'located'
P[SEM=''] -> 'in'
```

```
from nltk.parse import load_parser
cp = load_parser('grammars/book_grammars/sql0.fcfg')
```

```
query = 'What cities are located in China'
trees = list(cp.parse(query.split()))
print (trees)
```

```
[Tree(S[SEM=(SELECT, City FROM city_table, WHERE, , , Country="china")], [Tree(NP[SEM=(SELECT, City FROM city_table)], [Tree(De
t[SEM='SELECT'], ['What']), Tree(N[SEM='City FROM city_table'], ['cities'])]), Tree(VP[SEM=(, , Country="china")], [Tree(IV[SEM
=''], ['are']), Tree(AP[SEM=(, Country="china")], [Tree(A[SEM=''], ['located']), Tree(PP[SEM=(, Country="china")], [Tree(P[SEM
=''], ['in']), Tree(NP[SEM='Country="china"'], ['China'])])])])])]
```

```
answer = trees[0].label()['SEM']
print (answer)
answer = [s for s in answer if s]
```

```
(SELECT, City FROM city_table, WHERE, , , Country="china")
```

```
q = ' '.join(answer)
print(q)
```

```
SELECT City FROM city_table WHERE Country="china"
```


Roll No: Name:

```
from nltk.sem import chat80
nltk.download('city_database')
```

```
[nltk_data] Downloading package city_database to
[nltk_data]     C:\Users\lenovo\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\city_database.zip.
```

```
True
```

```
rows = chat80.sql_query('corpora/city_database/city.db', q)
```

```
for r in rows:
  print(r)
```

```
('canton',)
('chungking',)
('dairen',)
('harbin',)
('kowloon',)
('mukden',)
('peking',)
('shanghai',)
('sian',)
('tientsin',)
```


Roll No: Name:

Practical 6 b) **Date: 12/04/2023**

Aim:

b. Building a Discourse Representation Theory (DRT) by parsing a string representation - Angus owns a dog.

Code:

```
#pip install nltk

get_ipython().system('pip install nltk')

import nltk

read_the_expr = nltk.sem.DrtExpression.fromstring

drs1 = read_the_expr('([x, y], [Angus(x), dog(y), own(x, y)])')

print(drs1)

drs1.draw()

print(drs1.fol())

from nltk import load_parser

parser = load_parser('grammars/book_grammars/drt.fcfg',
logic_parser=nltk.sem.drt.DrtParser())

trees = list(parser.parse('Angus owns a dog'.split()))

print(trees[0].label()['SEM'].simplify())

trees[0].draw()
```

Output:

Roll No: Name:

```
import nltk
```

```
read_the_expr = nltk.sem.DrtExpression.fromstring
drs1 = read_the_expr('([x, y], [Angus(x), dog(y), own(x, y)])')
print(drs1)
drs1.draw()
```

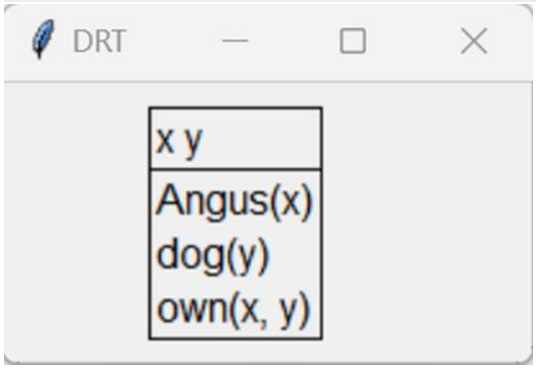
```
([x,y],[Angus(x), dog(y), own(x,y)])
```

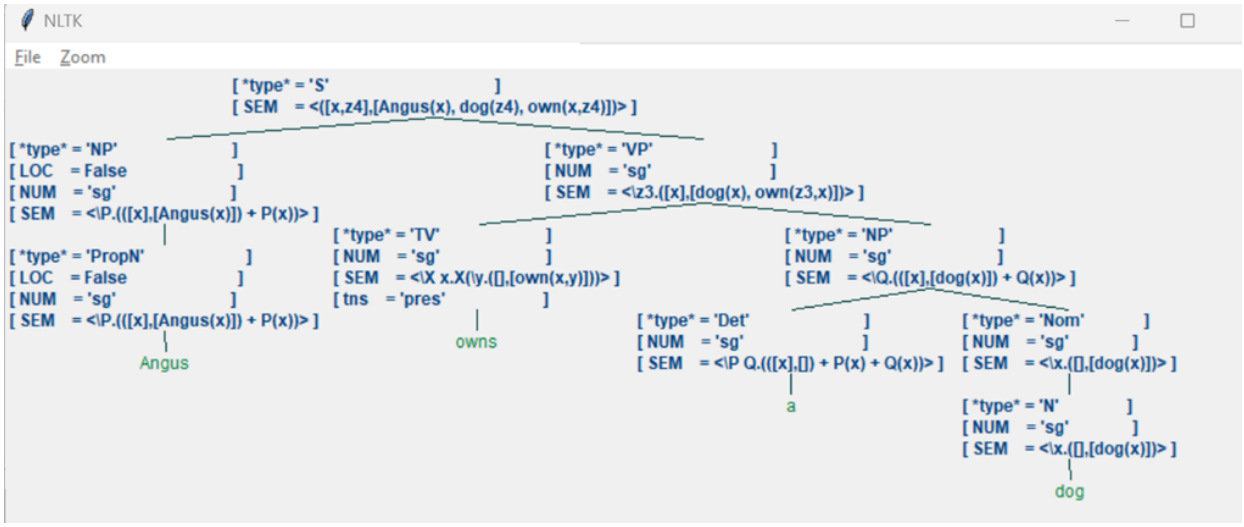
```
print(drs1.fol())
```

```
exists x y.(Angus(x) & dog(y) & own(x,y))
```

```
from nltk import load_parser
parser = load_parser('grammars/book_grammars/drt.fcfg', logic_parser=nltk.sem.drt.DrtParser())
trees = list(parser.parse('Angus owns a dog'.split()))
print(trees[0].label()['SEM'].simplify())
trees[0].draw()
```

```
([x,z2],[Angus(x), dog(z2), own(x,z2)])
```





Roll No: Name:

Practical No: 7 **Date: 14/04/2023**

Aim: Study PorterStemmer, LancasterStemmer, RegexpStemmer, SnowballStemmer and WordNetLemmatizer

Code:

PorterStemmer

```
import nltk
from nltk.stem import PorterStemmer
word_stemmer = PorterStemmer()
print(word_stemmer.stem('writing'))
```

#LancasterStemmer

```
import nltk
from nltk.stem import LancasterStemmer
Lanc_stemmer = LancasterStemmer()
print(Lanc_stemmer.stem('writing'))
```

#RegexpStemmer

```
import nltk
from nltk.stem import RegexpStemmer
Reg_stemmer = RegexpStemmer('ing$|s$|e$|able$', min=4)
print(Reg_stemmer.stem('writing'))
```

#SnowballStemmer

```
import nltk
from nltk.stem import SnowballStemmer
english_stemmer = SnowballStemmer('english')
print(english_stemmer.stem ('writing'))
```

#WordNetLemmatizer

```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
print("word :\tlemma")
print("rocks :", lemmatizer.lemmatize("rocks"))
print("corpora :", lemmatizer.lemmatize("corpora"))
# a denotes adjective in "pos"
print("better :", lemmatizer.lemmatize("better", pos ="a"))
```

Output:

```
word :	lemma
rocks : rock
corpora : corpus
better : good
```

```
import nltk
from nltk.stem import PorterStemmer
word_stemmer=PorterStemmer()
print(word_stemmer.stem('writing'))
```

```
write
```

```
#Lancaster Stemmer
from nltk.stem import LancasterStemmer
Lanc_Stemmer=LancasterStemmer()
print(Lanc_Stemmer.stem('writing'))
```

```
writ
```


Roll No: Name:

```
#RegexpStemmer
```

```
from nltk.stem import RegexpStemmer
```

```
Reg=RegexpStemmer('ing$|s$|e$|able$',min=4)
```

```
print(Reg.stem('writing'))
```

```
writ
```

```
#SnowballStemmer
```

```
from nltk.stem import SnowballStemmer
```

```
english_stemmer = SnowballStemmer('english')
```

```
print(english_stemmer.stem ('writing'))
```

```
write
```

```
#WordNetLemmatizer
```

```
from nltk.stem import WordNetLemmatizer
```

```
lemmatizer = WordNetLemmatizer()
```

```
print("word :\tlemma")
```

```
word :  lemma
```

```
print("rocks :", lemmatizer.lemmatize("rocks"))
```

```
rocks : rock
```

```
print("corpora :", lemmatizer.lemmatize("corpora"))
```

```
corpora : corpus
```

```
# a denotes adjective in "pos"
```

```
print("better :", lemmatizer.lemmatize("better", pos ="a"))
```

```
better : good
```


Roll No: Name:

Practical No: 8

Practical No: 8 a **Date: 14/04/2023**

Aim: a) Parse a sentence - "old men and women" and draw a tree using probabilistic parser

Code:

```
import nltk
from nltk import PCFG
grammar = PCFG.fromstring('''
NP -> NNS [0.5] | JJ NNS [0.3] | NP CC NP [0.2]
NNS -> "men" [0.1] | "women" [0.2] | "children" [0.3] | NNS CC NNS [0.4]
JJ -> "old" [0.4] | "young" [0.6]
CC -> "and" [0.9] | "or" [0.1]
''')
print(grammar)
viterbi_parser= nltk.ViterbiParser(grammar)
token = "old men and women".split()
obj = viterbi_parser.parse(token)
print("Output: ")
for x in obj:
    print(x)
    x.draw()
```

Output:

Roll No: Name:

In [1]:
```
import nltk
```

In [2]:
```
from nltk import PCFG
```

In [3]:
```
grammar = PCFG.fromstring('''
NP -> NNS [0.5] | JJ NNS [0.3] | NP CC NP [0.2]
NNS -> "men" [0.1] | "women" [0.2] | "children" [0.3] | NNS CC NNS [0.4]
JJ -> "old" [0.4] | "young" [0.6]
CC -> "and" [0.9] | "or" [0.1]
''')
```

In [5]:
```
print(grammar)
```

```
Grammar with 11 productions (start state = NP)
    NP -> NNS [0.5]
    NP -> JJ NNS [0.3]
    NP -> NP CC NP [0.2]
    NNS -> 'men' [0.1]
    NNS -> 'women' [0.2]
    NNS -> 'children' [0.3]
    NNS -> NNS CC NNS [0.4]
    JJ -> 'old' [0.4]
    JJ -> 'young' [0.6]
    CC -> 'and' [0.9]
    CC -> 'or' [0.1]
```

In [6]:
```
viterbi_parser= nltk.ViterbiParser(grammar)
```

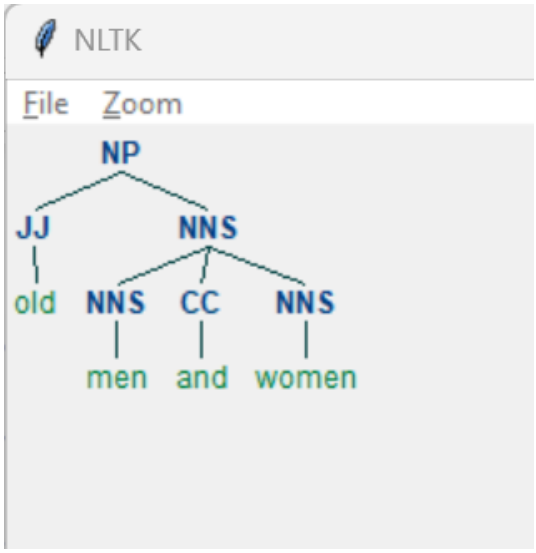
In [7]:
```
token = "old men and women".split()
```

In [8]:
```
obj = viterbi_parser.parse(token)
```

In [*]:
```
print("Output: ")
for x in obj:
    print(x)
    x.draw()
```

```
Output:
(NP (JJ old) (NNS (NNS men) (CC and) (NNS women))) (p=0.000864)
```


Roll No: Name:



Roll No: Name:

Practical No: 8 b **Date: 14/04/2023**

Aim: Parse a sentence - 'I saw a bird from my window.' and draw a tree using malt parsing.

HINT:

Set the environment variable -> System Variable -> New ->

Variable Name:(MALT-PARSER) -> Variable Value:(C:\Users\lenovo\AppData\Local\Programs\Python\Python310\maltparser-1.7.2)

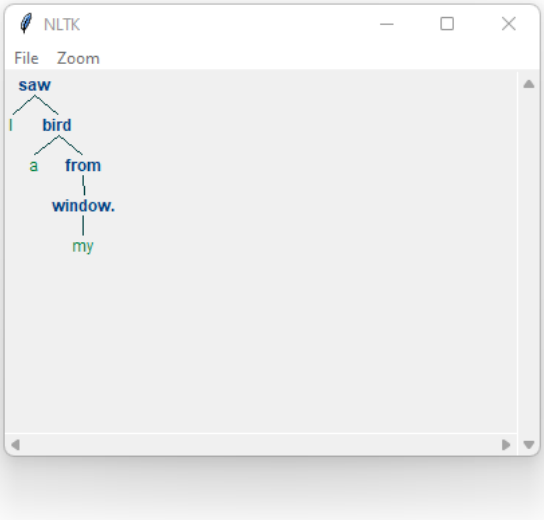
Variable Name:(MALT-MODEL) -> Variable Value:(C:\Users\lenovo\AppData\Local\Programs\Python\Python310\engmalt.linear-1.7.mco)

Code:

```
from nltk.parse import malt

mp = malt.MaltParser('maltparser-1.7.2','engmalt.linear-1.7.mco')

t = mp.parse_one('I saw a bird from my window.'.split()).tree()

print(t)

t.draw()
```

Output:

```
============== RESTART: C:\Users\admin\Deskto
(saw I (bird a (from (window. my))))
```



Roll No: Name:

Practical No: 9

Practical No: 9 a **Date: 14/04/2023**

Aim: Multiword Expressions in NLP for multiword – 'New Delhi' in '"Good cake cost Rs.1500\kg in New Delhi. Please buy me one of them.\n\nThanks."'

Code:

```
from nltk.tokenize import MWETokenizer

from nltk import sent_tokenize, word_tokenize

s = '"Good cake cost Rs.1500\kg in New Delhi. Please buy me one of them.\n\nThanks."'

mwe = MWETokenizer([('New','Delhi'), ('New','Bombay')], separator='_')

import nltk

nltk.download('punkt')

for sent in sent_tokenize(s):

    print(mwe.tokenize(word_tokenize(sent)))
```

Output:

```
In [1]: from nltk.tokenize import MWETokenizer

In [2]: from nltk import sent_tokenize, word_tokenize

In [3]: s = '''Good cake cost Rs.1500\kg in New Delhi. Please buy me one of them.\n\nThanks.'''

In [5]: mwe = MWETokenizer([('New','Delhi'), ('New','Bombay')], separator='_')

In [6]: import nltk

In [7]: nltk.download('punkt')
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\admin\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping tokenizers\punkt.zip.

Out[7]: True

In [8]: for sent in sent_tokenize(s):
            print(mwe.tokenize(word_tokenize(sent)))
['Good', 'cake', 'cost', 'Rs.1500\\kg', 'in', 'New_Delhi', '.']
['Please', 'buy', 'me', 'one', 'of', 'them', '.']
['Thanks', '.']
```


Roll No: Name:

Practical No: 9 b **Date: 14/04/2023**

Aim: Word Sense Disambiguation for the keyword ‘jam’ in the sentences - 'This device is used to jam the signal' and 'I am stuck in a traffic jam'. Also, for the keyword ‘book’ in the sentences - 'I love reading books on coding.' and 'The table was already booked by someone else.'

Code:

```
from nltk.wsd import lesk

from nltk.tokenize import word_tokenize

import nltk

nltk.download('omw-1.4')

nltk.download('punkt')

nltk.download('wordnet')

a1= lesk(word_tokenize('This device is used to jam the signal'),'jam')

print(a1,a1.definition())

a2 = lesk(word_tokenize('I am stuck in a traffic jam'),'jam')

print(a2,a2.definition())

b1= lesk(word_tokenize('I love reading books on coding.'),'book')

print(b1,b1.definition())

b2 = lesk(word_tokenize('The table was already booked by someone else.'),'book')

print(b2,b2.definition())
```

Output:

```
In [1]: from nltk.wsd import lesk
        from nltk.tokenize import word_tokenize

In [8]: import nltk
        nltk.download('omw-1.4')

        [nltk_data] Downloading package omw-1.4 to
        [nltk_data]     C:\Users\admin\AppData\Roaming\nltk_data...

Out[8]: True
```


Roll No: Name:

In [3]:
```
nltk.download('punkt')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\admin\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\admin\AppData\Roaming\nltk_data...
```

Out[3]: True

In [9]:
```
a1= lesk(word_tokenize('This device is used to jam the signal'),'jam')
print(a1,a1.definition())
a2 = lesk(word_tokenize('I am stuck in a traffic jam'),'jam')
print(a2,a2.definition())
```

```
Synset('jamming.n.01') deliberate radiation or reflection of electromagnetic energy for the purpose of disrupting enemy use of
electronic devices or systems
Synset('jam.v.05') get stuck and immobilized
```

In [10]:
```
b1= lesk(word_tokenize('I love reading books on coding.'),'book')
print(b1,b1.definition())
b2 = lesk(word_tokenize('The table was already booked by someone else.'),'book')
print(b2,b2.definition())
```

```
Synset('book.n.11') a number of sheets (ticket or stamps etc.) bound together on one edge
Synset('reserve.v.04') arrange for and reserve (something for someone else) in advance
```


Roll No: Name: