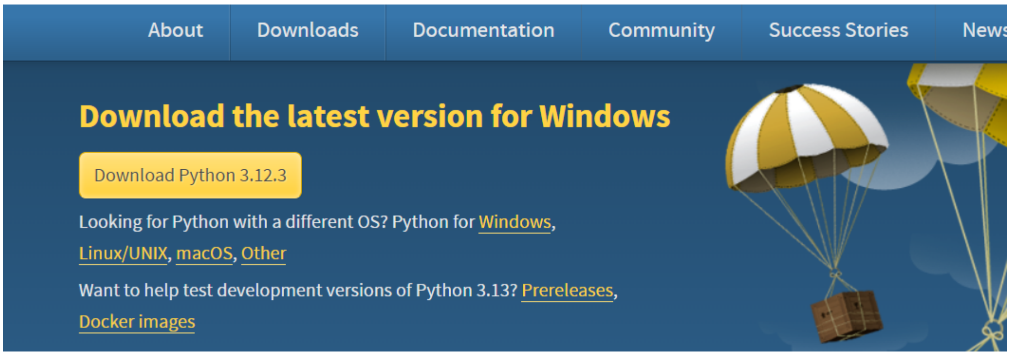
PRACTICAL 1

a) Install NLTK

Python 3.9.2 Installation on Windows

Step 1) Go to link https://www.python.org/downloads/, and select the latest

version for windows.

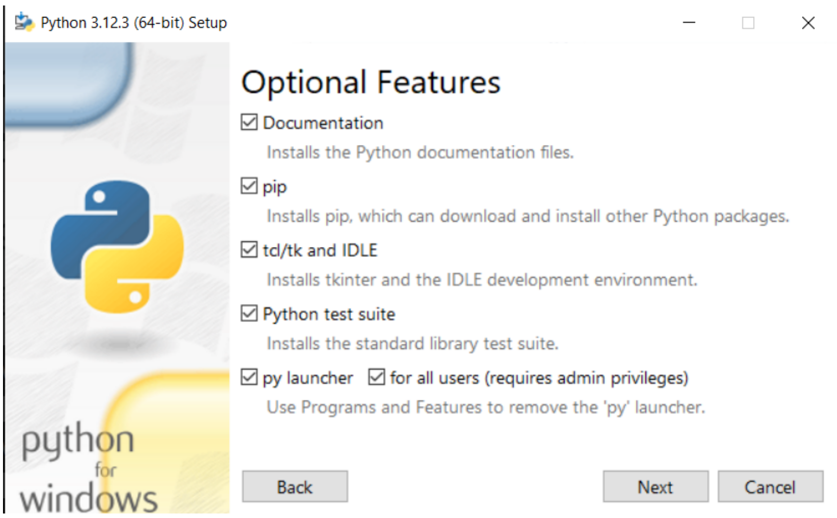


Note: If you don't want to download the latest version, you can visit the download tab and see all releases.

Step 2) Click on the Windows installer (64 bit)

Step 3) Select Customize Installation

Step 4) Click NEXT

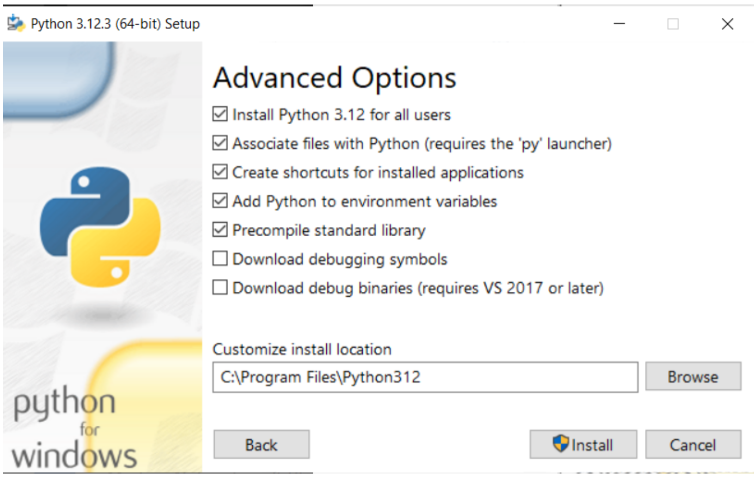


Step 5) In next screen

1. Select the advanced options

2. Give a Custom install location. Keep the default folder as c:\Program files\Python39

3. Click Install



Step 6) Click Close button once install is done.

Step 7) open command prompt window and run the following commands:

C:\Users\anike\AppData\Local\Programs\Python\Python310\Scripts>pip install --upgrade pip

C:\Users\anike\AppData\Local\Programs\Python\Python310\Scripts>pip install --user -U nltk

C:\Users\anike\AppData\Local\Programs\Python\Python310\Scripts>pip install --user -U numpy

C:\Users\anike\AppData\Local\Programs\Python\Python310\Scripts>python

>import nltk

b) Convert the given text to speech.

code:

```
# pip install gtts
 # pip install playsound
from playsound import playsound
# import required for text to speech conversion
 from gtts import gTTS
 mytext = "Welcome to Natural Language programming"
language = "en"
 myobj = gTTS(text=mytext, lang=language, slow=False)
 myobj.save("myfile.mp3")
 playsound("myfile.mp3")
```

Output:

welcomeNLP.mp3 audio file is getting created and it plays the file with playsound() method, while running the program.

c) Convert audio file Speech to Text.

Code:-

```
import speech_recognition as sr
 filename = “Greeting.wav"
 # initialize the recognizer
r = sr.Recognizer()
# open the file
        with sr.AudioFile(filename) as source:
         # listen for the data (load audio to memory)
        audio_data = r.record(source)
         # recognize (convert from speech to text)
         text = r.recognize_google(audio_data)
         print(text)
```

output:

```
good morning everyone
```

PRACTICAL 2

Aim:

a. Study of various Corpus – Brown, Inaugural, Reuters, udhr with various methods like fields,
raw, words, sents, categories,
b. Create and use your own corpora(plaintext, categorical)
c. Study Conditional frequency distributions Study of tagged corpora with methods like

tagged_sents, tagged_words.
d. Write a program to find the most frequent noun tags.
e. Map Words to Properties Using Python Dictionaries
f. Study DefaultTagger, Regular expression tagger, UnigramTagger
g. Find different words from a given plain text without any space by comparing this text with a given corpus of words. Also find the score of words.

a. **Study of various Corpus – Brown, Inaugural, Reuters, udhr with various methods like fields, raw, words, sents, categories,**
source code:

https://www.nltk.org/book/ch02.html
As just mentioned, a text corpus is a large body of text. Many corpora are designed to contain a careful balance of material in one or more genres. We examined some small text collections , such as the speeches known as the US Presidential Inaugural Addresses. This particular corpus actually contains dozens of individual texts — one per address — but for convenience we glued them end-to-end and treated them as a single text, also used various pre-defined texts that we accessed by typing from nltk.book import *. However, since we want to be able to work with other texts, this section examines a variety of text corpora. We'll see how to select individual texts, and how to work with them.

a.1 Brown Corpus
The Brown Corpus was the first million-word electronic corpus of English, created in 1961 at Brown University. This corpus contains text from 500 sources, and the sources have been categorized by genre, such as news, editorial, and so on. 1.1 gives an example of each genre.

Example Document for Each Section of the Brown Corpus

ID	File	Genre	Description
A16	ca16	news	Chicago Tribune: *Society Reportage*
B02	cb02	editorial	Christian Science Monitor: *Editorials*
C17	cc17	reviews	Time Magazine: *Reviews*
D12	cd12	religion	Underwood: *Probing the Ethics of Realtors*
E36	ce36	hobbies	Norling: *Renting a Car in Europe*
F25	cf25	lore	Boroff: *Jewish Teenage Culture*
G22	cg22	belles_lettres	Reiner: *Coping with Runaway Technology*
H15	ch15	government	US Office of Civil and Defence Mobilization: *The Family Fallout Shelter*
J17	cj19	learned	Mosteller: *Probability with Statistical Applications*
K04	ck04	fiction	W.E.B. Du Bois: *Worlds of Color*
L13	cl13	mystery	Hitchens: *Footsteps in the Night*
M01	cm01	science_fiction	Heinlein: *Stranger in a Strange Land*
N14	cn15	adventure	Field: *Rattlesnake Ridge*
P12	cp12	romance	Callaghan: *A Passion in Rome*
R06	cr06	humor	Thurber: *The Future, If Any, of Comedy*

We can access the corpus as a list of words, or a list of sentences (where each sentence is itself just a list of words). We can optionally specify particular categories or files to read:

```
import nltk
nltk.download('brown')
from nltk.corpus import brown
brown.categories()
```

```
[nltk_data] Downloading package brown to /root/nltk_data...
[nltk_data]   Unzipping corpora/brown.zip.
['adventure',
 'belles_lettres',
 'editorial',
 'fiction',
 'government',
 'hobbies',
 'humor',
 'learned',
 'lore',
 'mystery',
 'news',
 'religion',
 'reviews',
 'romance',
 'science_fiction']
```

```
[ ] brown.words(categories = 'news')
```

```
['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]
```

```
[ ] brown.fileids()
```

```
['ca01',
 'ca02',
 'ca03',
 'ca04',
 'ca05',
 'ca06',
 'ca07',
 'ca08',
 'ca09',
 'ca10',
 'ca11',
 'ca12',
 'ca13',
 'ca14',
 'ca15',
 'ca16',
 'ca17',
 'ca18',
```

```
[ ] brown.words(fileids='cg22')
```

```
['Does', 'our', 'society', 'have', 'a', 'runaway', ',', ...]
```

```
[ ] brown.sents(categories=['news','editorial','reviews'])
```

```
[['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', 'Friday', 'an', 'investigation', 'of', "Atlanta's", 'recent', 'primary', 'election', 'produced', '``', 'no', 'evidence', "''"
'that', 'any', 'irregularities', 'took', 'place', '.'], ['The', 'jury', 'further', 'said', 'in', 'term-end', 'presentments', 'that', 'the', 'City', 'Executive', 'Committee', ',',
'which', 'had', 'over-all', 'charge', 'of', 'the', 'election', ',', '``', 'deserves', 'the', 'praise', 'and', 'thanks', 'of', 'the', 'City', 'of', 'Atlanta', "''", 'for', 'the',
'manner', 'in', 'which', 'the', 'election', 'was', 'conducted', '.'], ...]
```

b. **Create and use your own corpora(plaintext, categorical)**

https://www.tutorialspoint.com/natural_language_toolkit/natural_language_toolkit_corpus_readers_and_custom_corpora.htm

How to build custom corpus?

While downloading NLTK, we also installed NLTK data package. So, we already have NLTK data package installed on our computer. If we talk about Windows, we'll assume that this data package is installed at C:\natural_language_toolkit_data and if we talk about Linux, Unix and

Mac OS X, we 'll assume that this data package is installed at /usr/share/natural_language_toolkit_data.
In the following Python recipe, we are going to create custom corpora which must be within one of the paths defined by NLTK. It is so because it can be found by NLTK. In order to avoid conflict with the official NLTK data package, let us create a custom natural_language_toolkit_data directory in our home directory.

```
[ ]  import os, os.path
     path = os.path.expanduser('~/natural_language_toolkit_data')
     if not os.path.exists(path):
       os.mkdir(path)
     os.path.exists(path)
```

```
True
```

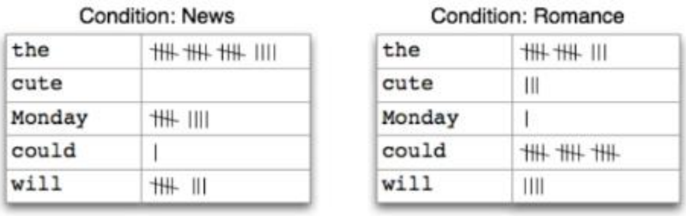
Now we will make a wordlist file, named wordfile.txt and put it in a folder, named corpus in nltk_data directory (content/corpus/wordfile.txt) and will load it by using nltk.data.load –
Corpus readers
NLTK provides various CorpusReader classes. We are going to cover them in the following python recipes

Creating wordlist corpus
NLTK has WordListCorpusReader class that provides access to the file containing a list of words. For the following Python recipe, we need to create a wordlist file which can be CSV or normal text file. For example, we have created a file named 'list' that contains the following data –

```
[3]  from nltk.corpus.reader import WordListCorpusReader
     reader_corpus = WordListCorpusReader('.',['/wordfile.txt'])
     reader_corpus.words()
```

```
['Hello', 'this', 'is', 'Practical', 'no', '2']
```

C. **Study Conditional frequency distributions**
https://www.nltk.org/book/ch02.html
Conditional Frequency Distributions
When the texts of a corpus are divided into several categories, by genre, topic, author, etc, we can maintain separate frequency distributions for each category. This will allow us to study systematic differences between the categories. In the previous section we achieved this using NLTK's ConditionalFreqDist data type. A conditional frequency distribution is a collection of frequency distributions, each one for a different "condition". The condition will often be the category of the text. 2.1 depicts a fragment of a conditional frequency distribution having just two conditions, one for news text and one for romance text.



2.1 **Conditions and Events**
A frequency distribution counts observable events, such as the appearance of words in a text. A conditional frequency distribution needs to pair each event with a condition. So instead of processing a sequence of words ,
we have to process a sequence of pairs :
text = ['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]
pairs = [('news', 'The'), ('news', 'Fulton'), ('news', 'County'), ...]
Each pair has the form (condition, event). If we were processing the entire Brown Corpus by genre there would be 15 conditions (one per genre), and 1,161,192 events (one per word).

2.2 Counting Words by Genre
In 1 we saw a conditional frequency distribution where the condition was the section of the Brown Corpus, and for each condition we counted words. Whereas FreqDist() takes a simple list as input, ConditionalFreqDist() takes a list of pairs.

```
>>> from nltk.corpus import brown
>>> cfd = nltk.ConditionalFreqDist(
... (genre, word)
... for genre in brown.categories()
... for word in brown.words(categories=genre))
```

Let's break this down, and look at just two genres, news and romance. For each genre , we loop over every word in the genre , producing pairs consisting of the genre and the word
:

```
>>> genre_word = [(genre, word) [1]
... for genre in ['news', 'romance'] [2]
... for word in brown.words(categories=genre)] [3]
>>> len(genre_word)
```

```
genre_word = [(genre, word)
      for genre in ['news', 'romance']
      for word in brown.words(categories=genre)]
print(len(genre_word))
```

```
[nltk_data] Downloading package inaugural to /root/nltk_data...
[nltk_data]   Package inaugural is already up-to-date!
[nltk_data] Downloading package udhr to /root/nltk_data...
[nltk_data]   Package udhr is already up-to-date!
[nltk_data] Downloading package brown to /root/nltk_data...
[nltk_data]   Unzipping corpora/brown.zip.
170576
```

So, as we can see below, pairs at the beginning of the list genre_word will be of the form ('news', word) , while those at the end will be of the form ('romance', word) .

```
print(genre_word[:4])
```

```
[('news', 'The'), ('news', 'Fulton'), ('news', 'County'), ('news', 'Grand')]
```

+ Code + Text

```
print(genre_word[-4:])
```

```
[('romance', 'afraid'), ('romance', 'not'), ('romance', "''"), ('romance', '.')]
```

Study of tagged corpora with methods like tagged_sents, tagged_words.
http://www.nltk.org/book/ch05.html
The process of classifying words into their parts of speech and labeling them accordingly is known as part-of-speech tagging, POS-tagging, or simply tagging. Parts of speech are also known as word classes or lexical categories. The collection of tags used for a particular task is known as a tagset. Our emphasis in this chapter is on exploiting tags, and tagging text automatically.

1 Using a Tagger

A part-of-speech tagger, or POS-tagger, processes a sequence of words, and attaches a part of speech tag to each word (don't forget to import nltk):

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

para = "And now we are going to learn something new"
sents = tokenize.sent_tokenize(para)
print("\nsentence tokenization\n===================\n",sents)

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

```
sentence tokenization
===================
 ['And now we are going to learn something new']

word tokenization
===================

[('And', 'CC'), ('now', 'RB'), ('we', 'PRP'), ('are', 'VBP'), ('going', 'VBG'), ('to', 'TO'), ('learn', 'VB'), ('something', 'NN'), ('new', 'JJ'
```

Other corpora use a variety of formats for storing part-of-speech tags. NLTK's corpus readers provide a uniform interface so that you don't have to be concerned with the different file formats. In contrast with the file fragment shown above, the corpus reader for the Brown Corpus represents the data as shown below. Note that part-of-speech tags have been converted to uppercase, since this has become standard practice since the Brown Corpus was published.

```
import nltk
nltk.corpus.brown.tagged_words()
```

```
[('The', 'AT'), ('Fulton', 'NP-TL'), ...]
```

```
import nltk
nltk.download('universal_tagset')
nltk.corpus.brown.tagged_words(tagset='universal')
```

```
[nltk_data] Downloading package universal_tagset to /root/nltk_data...
[nltk_data]   Unzipping taggers/universal_tagset.zip.
[('The', 'DET'), ('Fulton', 'NOUN'), ...]
```

Whenever a corpus contains tagged text, the NLTK corpus interface will have a tagged_words() method. Here are some more examples, again using the output format illustrated for the Brown Corpus:

```
import nltk
nltk.download('nps_chat')
print(nltk.corpus.nps_chat.tagged_words())
```

```
[nltk_data] Downloading package nps_chat to /root/nltk_data...
[nltk_data]   Unzipping corpora/nps_chat.zip.
[('now', 'RB'), ('im', 'PRP'), ('left', 'VBD'), ...]
```

```
import nltk
nltk.download('conll2000')
nltk.corpus.conll2000.tagged_words()
```

```
[nltk_data] Downloading package conll2000 to /root/nltk_data...
[nltk_data]   Unzipping corpora/conll2000.zip.
[('Confidence', 'NN'), ('in', 'IN'), ('the', 'DT'), ...]
```

```
import nltk
nltk.download('treebank')
nltk.corpus.treebank.tagged_words()
```

```
[nltk_data] Downloading package treebank to /root/nltk_data...
[nltk_data]   Unzipping corpora/treebank.zip.
[('Pierre', 'NNP'), ('Vinken', 'NNP'), (',', ','), ...]
```

Automatic Tagging

In the rest of this chapter we will explore various ways to automatically add part-of-speech tags to text. We will see that the tag of a word depends on the word and its context within a sentence. For this reason, we will be working with data at the level of (tagged) sentences rather than words. We'll begin by loading the data we will be using.

The Default Tagger

The simplest possible tagger assigns the same tag to each token. This may seem to be a rather banal step, but it establishes an important baseline for tagger performance. In order to get the best result, we tag each word with the most likely tag. Let's find out which tag is most likely (now using the unsimplified tagset):

```
from nltk.corpus import brown
brown_tagged_sents = brown.tagged_sents(categories='news')
brown_sents = brown.sents(categories='news')
tags = [tag for (word, tag) in brown.tagged_words(categories='news')]
nltk.FreqDist(tags).max()
```

```
'NN'
```

d. Write a program to find the most frequent noun tags.

```
nltk.download('universal_tagset')
from nltk.corpus import brown
noundist = nltk.FreqDist(w2 for ((w1, t1), (w2, t2)) in
      nltk.bigrams(brown.tagged_words(tagset="universal"))
      if w1.lower() == "the" and t2 == "NOUN")
noundist.most_common(10)
```

```
[nltk_data] Downloading package universal_tagset to /root/nltk_data...
[nltk_data]   Unzipping taggers/universal_tagset.zip.
[('world', 346),
 ('time', 250),
 ('way', 236),
 ('end', 206),
 ('fact', 194),
 ('state', 190),
 ('man', 176),
 ('door', 172),
 ('house', 152),
 ('city', 127)]
```

e. Map Words to Properties Using Python Dictionaries

```
#creating and printing a dictionay by mapping word with its properties
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
print(thisdict["brand"])
print('length:', len(thisdict))
print(type(thisdict))
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
Ford
length: 3
<class 'dict'>
```

f. Study DefaultTagger, Regular expression tagger, UnigramTagger

A. DefaultTagger

Program:

```
[13] from nltk.corpus import brown
     nltk.download('brown')
     nltk.download('punkt')
     tags = [tag for (word, tag) in brown.tagged_words(categories='news')]
     nltk.FreqDist(tags).max()
     raw = 'I do not like green eggs and ham, I do not like them Sam I am!'
     tokens = nltk.word_tokenize(raw)
     default_tagger = nltk.DefaultTagger('NN')
     default_tagger.tag(tokens)
```

```
[nltk_data] Downloading package brown to /root/nltk_data...
[nltk_data]   Package brown is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[('I', 'NN'),
 ('do', 'NN'),
 ('not', 'NN'),
 ('like', 'NN'),
 ('green', 'NN'),
 ('eggs', 'NN'),
 ('and', 'NN'),
 ('ham', 'NN'),
 (',', 'NN'),
 ('I', 'NN'),
 ('do', 'NN'),
 ('not', 'NN'),
 ('like', 'NN'),
 ('them', 'NN'),
 ('Sam', 'NN'),
```

B. Regular Expression Tagger:

```
import nltk
nltk.download('brown')
nltk.download('punkt')
from nltk.corpus import brown
from nltk import word_tokenize
from nltk import RegexpTagger
brown_sents = brown.sents(categories = 'news')
brown_tagged_sents = brown.tagged_sents(categories = 'news')
import nltk
nltk.download('brown')
nltk.download('punkt')
from nltk.corpus import brown
from nltk import word_tokenize
from nltk import RegexpTagger
brown_sents = brown.sents(categories = 'news')
brown_tagged_sents = brown.tagged_sents(categories = 'news')
patterns = [
(r'.*ing$', 'VBG'), # gerunds
(r'.*ed$', 'VBD'), # simple past
(r'.*es$', 'VBZ'), # 3rd singular present
(r'.*ould$', 'MD'), # modals
(r'.*\'s$', 'NN$'), # possessive nouns
(r'.*s$', 'NNS'), # plural nouns
(r'^-?[0-9]+(\.[0-9]+)?$', 'CD'), # cardinal numbers
(r'.*', 'NN') # nouns (default)
]
regexp_tagger = nltk.RegexpTagger(patterns)
regexp_tagger.tag(brown_sents[3])
```

```
[nltk_data]   Package brown is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[('``', 'NN'),
 ('Only', 'NN'),
 ('a', 'NN'),
 ('relative', 'NN'),
 ('handful', 'NN'),
 ('of', 'NN'),
 ('such', 'NN'),
 ('reports', 'NNS'),
 ('was', 'NNS'),
 ('received', 'VBD'),
 ("''", 'NN'),
 (',', 'NN'),
 ('the', 'NN'),
 ('jury', 'NN'),
 ('said', 'NN'),
 (',', 'NN'),
 ('``', 'NN'),
 ('considering', 'VBG'),
 ('the', 'NN'),
 ('widespread', 'NN'),
 ('interest', 'NN'),
 ('in', 'NN'),
 ('the', 'NN'),
 ('election', 'NN'),
 (',', 'NN'),
 ('the', 'NN'),
 ('number', 'NN'),
 ('of', 'NN'),
 ('voters', 'NNS'),
 ('and', 'NN'),
```

C. Unigram Tagger

```
import nltk
nltk.download('brown')
nltk.download('punkt')
from nltk.corpus import brown
from nltk import UnigramTagger
brown_tagged_sents = brown.tagged_sents(categories = 'news')
brown_sents = brown.sents(categories = 'news')
unigram_tagger = nltk.UnigramTagger(brown_tagged_sents)
tags = unigram_tagger.tag(brown_sents[2007])
print("\nDisplaying the tags from brown sents : \n", tags)
evaluation = unigram_tagger.evaluate(brown_tagged_sents)
print("\nEvaluation of brown tagged sents : ", evaluation)
```

```
[nltk_data] Downloading package brown to /root/nltk_data...
[nltk_data]   Package brown is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!

Displaying the tags from brown sents :
 [('Various', 'JJ'), ('of', 'IN'), ('the', 'AT'), ('apartments', 'NNS'), ('are', 'BER'), ('of', 'IN'), ('the', 'AT'), ('terrace', 'NN'), ('type'
<ipython-input-34-3468a80f160a>:11: DeprecationWarning:
  Function evaluate() has been deprecated.  Use accuracy(gold)
  instead.
  evaluation = unigram_tagger.evaluate(brown_tagged_sents)

Evaluation of brown tagged sents :  0.9349006503968017
```

PRACTICAL 3

Aim:-

a. Study of Wordnet Dictionary with methods as synsets, definitions, examples, antonyms.
b. Study lemmas, hyponyms, hypernyms, entailments,
c. Write a program using python to find synonym and antonym of word "active" using Wordnet
d. Compare two nouns
e. Handling stopword.
Using nltk Adding or Removing Stop Words in NLTK's Default Stop Word List
Using Gensim Adding and Removing Stop Words in Default Gensim Stop Words List
Using Spacy Adding and Removing Stop Words in Default Spacy Stop Words List

a. Study of Wordnet Dictionary with methods as synsets, definitions, examples, antonyms.

A. Synset()

WordNet is the lexical database i.e. dictionary for the English language, specifically designed for natural language processing.

Synset is a special kind of a simple interface that is present in NLTK to look up words in WordNet. Synset instances are the groupings of synonymous words that express the same concept. Some of the words have only one Synset and some have several.

Output:

```
[Synset('computer.n.01'), Synset('calculator.n.01')]
a machine for performing calculations automatically
Examples: []
[Lemma('sell.v.01.sell')]
```

b. Study lemmas, hyponyms, hypernyms.

Code:

```
import nltk
from nltk.corpus import wordnet
print(wordnet.synsets("computer"))
print(wordnet.synset("computer.n.01").lemma_names())
#all lemmas for each synset.
for e in wordnet.synsets("computer"):
print(f'{e} --> {e.lemma_names()}')
#print all lemmas for a given synset
print(wordnet.synset('computer.n.01').lemmas())
#get the synset corresponding to lemma
print(wordnet.lemma('computer.n.01.computing_device').synset())
```

```
#Get the name of the lemma
print(wordnet.lemma('computer.n.01.computing_device').name())
#Hyponyms give abstract concepts of the word that are much more specific
#the list of hyponyms words of the computer
syn = wordnet.synset('computer.n.01')
print(syn.hyponyms)
print([lemma.name() for synset in syn.hyponyms() for lemma in synset.lemmas()])
#the semantic similarity in WordNet
vehicle = wordnet.synset('vehicle.n.01')
car = wordnet.synset('car.n.01')
print(car.lowest_common_hypernyms(vehicle))
```

Output:

```
[Synset('computer.n.01'), Synset('calculator.n.01')]
['computer', 'computing_machine', 'computing_device', 'data_processor', 'electronic_computer', 'information_processing_system']
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']
[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')]
Synset('computer.n.01')
computing_device
<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']
[Synset('vehicle.n.01')]
```

c. Write a program using python to find synonym and antonym of word "active" using Wordnet.

Code:

```
from nltk.corpus import wordnet
print( wordnet.synsets("active"))
print(wordnet.lemma('active.a.01.active').antonyms())
```

Output:

```
[Synset('active_agent.n.01'), Synset('active_voice.n.01'), Synset('active.n.03'), Synset('active.a.01'), Synset('active.s.02'),
Synset('active.a.03'), Synset('active.s.04'), Synset('active.a.05'), Synset('active.a.06'), Synset('active.a.07'), Synset('acti
ve.s.08'), Synset('active.a.09'), Synset('active.a.10'), Synset('active.a.11'), Synset('active.a.12'), Synset('active.a.13'), S
ynset('active.a.14')]
[Lemma('inactive.a.02.inactive')]
```

d. Compare two nouns

Code:

```
import nltk
from nltk.corpus import wordnet
syn1 = wordnet.synsets('football')
syn2 = wordnet.synsets('soccer')
# A word may have multiple synsets, so need to compare each synset of word1
with synset of word2
for s1 in syn1:
for s2 in syn2:
print("Path similarity of: ")
print(s1, '(', s1.pos(), ')', '[', s1.definition(), ']')
```

```
print(s2, '(', s2.pos(), ')', '[', s2.definition(), ']')
print(" is", s1.path_similarity(s2))
print()
```

Output:

```
Path similarity of:
Synset('football.n.01') ( n ) [ any of various games played with a ball (round or oval) in which two teams try to kick or carry
or propel the ball into each other's goal ]
Synset('soccer.n.01') ( n ) [ a football game in which two teams of 11 players try to kick or head a ball into the opponents' g
oal ]
 is 0.5

Path similarity of:
Synset('football.n.02') ( n ) [ the inflated oblong ball used in playing American football ]
Synset('soccer.n.01') ( n ) [ a football game in which two teams of 11 players try to kick or head a ball into the opponents' g
oal ]
 is 0.05
```

e. Handling stopword:

i) Using nltk Adding or Removing Stop Words in NLTK's Default Stop Word List

```
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
from nltk.tokenize import word_tokenize
text = "Yashesh likes to play football, however he is not too fond of tennis."
text_tokens = word_tokenize(text)
tokens_without_sw = [word for word in text_tokens if not word in
stopwords.words()]
print(tokens_without_sw)
#add the word play to the NLTK stop word collection
all_stopwords = stopwords.words('english')
all_stopwords.append('play')
text_tokens = word_tokenize(text)
tokens_without_sw = [word for word in text_tokens if not word in all_stopwords]
print(tokens_without_sw)
#remove ‘not’ from stop word collection
all_stopwords.remove('not')
text_tokens = word_tokenize(text)
tokens_without_sw = [word for word in text_tokens if not word in all_stopwords]
print(tokens_without_sw)
```

Output:

```
['Yashesh', 'likes', 'play', 'football', ',', 'fond', 'tennis', '.']
['Yashesh', 'likes', 'football', ',', 'however', 'fond', 'tennis', '.']
['Yashesh', 'likes', 'football', ',', 'however', 'not', 'fond', 'tennis', '.']
```

ii) Using Gensim Adding and Removing Stop Words in Default Gensim Stop Words List

Code:

```
#pip install gensim
import gensim
from gensim.parsing.preprocessing import remove_stopwords
text = "Yashesh likes to play football, however he is not too fond of tennis."
filtered_sentence = remove_stopwords(text)
print(filtered_sentence)
all_stopwords = gensim.parsing.preprocessing.STOPWORDS
print(all_stopwords)
'''The following script adds likes and play to the list of stop words in Gensim:'''
from gensim.parsing.preprocessing import STOPWORDS
all_stopwords_gensim = STOPWORDS.union(set(['likes', 'play']))
text = "Yashesh likes to play football, however he is not too fond of tennis."
text_tokens = word_tokenize(text)
tokens_without_sw = [word for word in text_tokens if not word in
all_stopwords_gensim]
print(tokens_without_sw)
'''Output:
['Yashesh', 'football', ',', 'fond', 'tennis', '.']
The following script removes the word "not" from the set of stop words in
Gensim:'''
from gensim.parsing.preprocessing import STOPWORDS
all_stopwords_gensim = STOPWORDS
sw_list = {"not"}
all_stopwords_gensim = STOPWORDS.difference(sw_list)
text = "Yashesh likes to play football, however he is not too fond of tennis."
text_tokens = word_tokenize(text)
tokens_without_sw = [word for word in text_tokens if not word in
all_stopwords_gensim]
print(tokens_without_sw)
```

Output:

```
Messi likes play football, fond tennis.
frozenset({'meanwhile', 'move', 'others', 'don', 'whom', 'so', 'nor', 'whereafter', 'put', 'the', 'herein', 'well', 'will', 'fr
om', 'everything', 're', 'alone', 'few', 'whatever', 'always', 'elsewhere', 'almost', 'sixty', 'eight', 'get', 'go', 'wherein',
'both', 'onto', 'again', 'part', 'kg', 'also', 'via', 'a', 'no', 'something', 'our', 'call', 'six', 'two', 'twelve', 'until',
'etc', 'somewhere', 'she', 'seems', 'amongst', 'too', 'it', 'if', 'those', 'after', 'yet', 'nowhere', 'themselves', 'detail',
'to', 'whereas', 'they', 'mill', 'see', 'forty', 'over', 'hundred', 'please', 'none', 'below', 'find', 'were', 'full', 'them',
'most', 'became', 'nothing', 'hereupon', 'where', 'such', 'all', 'only', 'ltd', 'never', 'mostly', 'hence', 'am', 'hasnt', 'fiv
e', 'their', 'give', 'fifteen', 'less', 'against', 'co', 'now', 'and', 'bottom', 'seemed', 'least', 'by', 'within', 'toward',
'may', 'first', 'ie', 'would', 'anywhere', 'hereafter', 'doesn', 'as', 'when', 'sometime', 'using', 'across', 'an', 'besides',
'anyhow', 'interest', 'of', 'around', 'everyone', 'own', 'into', 'unless', 'behind', 'every', 'have', 'several', 'km', 'thereb
y', 'how', 'didn', 'bill', 'whereby', 'for', 'latter', 'otherwise', 'often', 'wherever', 'amount', 'any', 'must', 'together',
'is', 'name', 'front', 'become', 'then', 'ourselves', 'might', 'cant', 'sometimes', 'although', 'nobody', 'amoungst', 'througho
ut', 'con', 'that', 'rather', 'above', 'ever', 'whither', 'neither', 'serious', 'thereafter', 'did', 'has', 'however', 'formerl
y', 'seem', 'does', 'thus', 'eg', 'used', 'made', 'me', 'last', 'with', 'could', 'was', 'mine', 'while', 'been', 'becoming', 'e
verywhere', 'anything', 'some', 'off', 'during', 'whose', 'other', 'various', 'seeming', 'whenever', 'third', 'once', 'be', 'th
erein', 'say', 'this', 'what', 'between', 'un', 'its', 'afterwards', 'else', 'you', 'under', 'your', 'here', 'more', 'each', 'd
escribe', 'moreover', 'quite', 'beyond', 'i', 'about', 'sincere', 'beforehand', 'done', 'or', 'yourself', 'without', 'becomes',
'four', 'yourselves', 'latterly', 'yours', 'found', 'per', 'whoever', 'already', 'noone', 'whereupon', 'her', 'cry', 'make', 'm
uch', 'had', 'thereupon', 'him', 'we', 'empty', 'show', 'ten', 'his', 'who', 'hereby', 'really', 'thence', 'anyone', 'us', 'the
refore', 'but', 'among', 'these', 'on', 'further', 'thick', 'himself', 'eleven', 'one', 'do', 'towards', 'along', 'except', 'pe
rhaps', 'very', 'indeed', 'down', 'before', 'side', 'next', 'even', 'whence', 'upon', 'in', 'fill', 'same', 'fire', 'my', 'thr
u', 'than', 'someone', 'not', 'can', 'ours', 'beside', 'herself', 'since', 'there', 'nine', 'de', 'enough', 'at', 'still', 'fif
ty', 'out', 'whole', 'doing', 'former', 'though', 'being', 'system', 'take', 'thin', 'keep', 'inc', 'hers', 'myself', 'just',
'many', 'nevertheless', 'are', 'twenty', 'due', 'either', 'anyway', 'through', 'up', 'whether', 'why', 'back', 'itself', 'namel
y', 'three', 'cannot', 'should', 'another', 'he', 'couldnt', 'somehow', 'regarding', 'because', 'computer', 'top', 'which'})
['Messi', 'football', ',', 'fond', 'tennis', '.']
['Yashesh', 'likes', 'play', 'football', ',', 'not', 'fond', 'tennis', '.']
```

 # Remove the word 'not' from the existing set of Gensim stop words

Code:-

```
from gensim.parsing.preprocessing import STOPWORDS
all_stopwords_gensim = STOPWORDS
sw_list = {"not"}
all_stopwords_gensim = STOPWORDS.difference(sw_list)
text = "Yashesh likes to play football, however he is not too fond of tennis."
text_tokens = word_tokenize(text)
# Filter out the tokens again with 'not' removed from stop words set
tokens_without_sw = [word for word in text_tokens if not word in
all_stopwords_gensim]
print(tokens_without_sw)
```

Output:-

```
['Yashesh', 'likes', 'play', 'football', ',', 'not', 'fond', 'tennis', '.']
```

iii) Using Spacy Adding and Removing Stop Words in Default Spacy Stop Words List

code:-

```
#pip install spacy
#python -m spacy download en_core_web_sm
#python -m spacy download en
import spacy
import nltk
from nltk.tokenize import word_tokenize
sp = spacy.load('en_core_web_sm')
#add the word play to the NLTK stop word collection
all_stopwords = sp.Defaults.stop_words
all_stopwords.add("play")
text = "Yashesh likes to play football, however he is not too fond of tennis."
text_tokens = word_tokenize(text)
tokens_without_sw = [word for word in text_tokens if not word in all_stopwords]
print(tokens_without_sw)
#remove 'not' from stop word collection
all_stopwords.remove('not')
tokens_without_sw = [word for word in text_tokens if not word in all_stopwords]
print(tokens_without_sw)
```

Output:

```
['Yashesh', 'likes', 'football', ',', 'fond', 'tennis', '.']
['Yashesh', 'likes', 'football', ',', 'not', 'fond', 'tennis', '.']
```

PRACTICAL 4

Text Tokenization

a. Tokenization using Python's split() function

code:

```
text = """Founded in 2002, SpaceX's mission is to enable humans to become a
spacefaring civilization and a multi-
planet

species by building a self-sustaining city on Mars. In 2008, SpaceX's Falcon 1 became the
first privately developed
liquid-fuel launch vehicle to orbit the Earth."""
# Splits at space
a=text.split()
print(a)
```

Output:

```
['Founded', 'in', '2002,', 'SpaceX's', 'mission', 'is', 'to', 'enable', 'humans', 'to', 'become', 'a', 'spacefaring', 'civiliza
tion', 'and', 'a', 'multi-', 'planet', 'species', 'by', 'building', 'a', 'self-sustaining', 'city', 'on', 'Mars.', 'In', '200
8,', 'SpaceX's', 'Falcon', '1', 'became', 'the', 'first', 'privately', 'developed', 'liquid-fuel', 'launch', 'vehicle', 'to',
'orbit', 'the', 'Earth.']
```

4 b): Tokenization using Regular Expressions (RegEx)

Code:-

```
import re

text = """Founded in 2002, SpaceX's mission is to enable humans to become a
spacefaring civilization and a multi-
planet

species by building a self-sustaining city on Mars. In 2008, SpaceX's Falcon 1 became the
first privately developed
liquid-fuel launch vehicle to orbit the Earth."""
tokens = re.findall("[\w']+", text)
print(tokens)
```

Output:

```
['Founded', 'in', '2002', 'SpaceX', 's', 'mission', 'is', 'to', 'enable', 'humans', 'to', 'become', 'a', 'spacefaring', 'civili
zation', 'and', 'a', 'multi', 'planet', 'species', 'by', 'building', 'a', 'self', 'sustaining', 'city', 'on', 'Mars', 'In', '20
08', 'SpaceX', 's', 'Falcon', '1', 'became', 'the', 'first', 'privately', 'developed', 'liquid', 'fuel', 'launch', 'vehicle',
'to', 'orbit', 'the', 'Earth']
```

4 c): Tokenization using NLTK

Code:-

```
import nltk

nltk.download('punkt')
from nltk.tokenize import word_tokenize

text = """Founded in 2002, SpaceX’s mission is to enable humans to become a
spacefaring civilization and a multi-
planet

species by building a self-sustaining city on Mars. In 2008, SpaceX’s Falcon 1 became the
first privately developed
liquid-fuel launch vehicle to orbit the Earth."""
a=word_tokenize(text)
print(a)
```

Output:

```
['Founded', 'in', '2002', ',', 'SpaceX', '’', 's', 'mission', 'is', 'to', 'enable', 'humans', 'to', 'become', 'a', 'spacefarin
g', 'civilization', 'and', 'a', 'multi-', 'planet', 'species', 'by', 'building', 'a', 'self-sustaining', 'city', 'on', 'Mars',
'.', 'In', '2008', ',', 'SpaceX', '’', 's', 'Falcon', '1', 'became', 'the', 'first', 'privately', 'developed', 'liquid-fuel',
'launch', 'vehicle', 'to', 'orbit', 'the', 'Earth', '.']
```

4 d) Tokenization using the spaCy library

```
#pip install -U spacy
#python -m spacy download en
from spacy.lang.en import English
# Load English tokenizer, tagger, parser, NER and word vectors
nlp = English()
text = """Founded in 2002, SpaceX’s mission is to enable humans to become a
spacefaring civilization and a multi-planet
species by building a self-sustaining city on Mars. In 2008, SpaceX’s Falcon 1 became the
first privately developed
liquid-fuel launch vehicle to orbit the Earth."""
# "nlp" Object is used to create documents with linguistic annotations.
my_doc = nlp(text)
# Create list of word tokens
token_list = []
for token in my_doc:
token_list.append(token.text)
```

Natural Language Processing

```
token_list
print(token_list)
```

Output:

```
['Founded', 'in', '2002', ',', 'SpaceX', ''s', 'mission', 'is', 'to', 'enable', 'humans', 'to', 'become', 'a', 'spacefaring',
'civilization', 'and', 'a', 'multi', '-', 'planet', '\n', 'species', 'by', 'building', 'a', 'self', '-', 'sustaining', 'city',
'on', 'Mars', '.', 'In', '2008', ',', 'SpaceX', ''s', 'Falcon', '1', 'became', 'the', 'first', 'privately', 'developed', '\n',
'liquid', '-', 'fuel', 'launch', 'vehicle', 'to', 'orbit', 'the', 'Earth', '.']
```

4 e) Tokenization using Keras.

Code:-

```
from keras.preprocessing.text import text_to_word_sequence
# define
text = """Founded in 2002, SpaceX’s mission is to enable humans to become a spacefa
ring civilization and a multi-planet

species by building a self-
sustaining city on Mars. In 2008, SpaceX’s Falcon 1 became the first privately deve

loped
liquid-fuel launch vehicle to orbit the Earth."""
# tokenize
result = text_to_word_sequence(text)
print(result)
```

Output:

```
['founded', 'in', '2002', 'spacex's', 'mission', 'is', 'to', 'enable', 'humans', 'to', 'become', 'a', 'spacefa', 'ring', 'civil
ization', 'and', 'a', 'multi', 'planet', 'species', 'by', 'building', 'a', 'self', 'sustaining', 'city', 'on', 'mars', 'in', '2
008', 'spacex's', 'falcon', '1', 'became', 'the', 'first', 'privately', 'deve', 'loped', 'liquid', 'fuel', 'launch', 'vehicle',
'to', 'orbit', 'the', 'earth']
```

4 f) Tokenization using Gensim.

Code:-

```
from gensim.utils import tokenize
text = """Founded in 2002, SpaceX’s mission is to enable humans to become a spacefa
ring civilization and a multi-planet

species by building a self-
sustaining city on Mars. In 2008, SpaceX’s Falcon 1 became the first privately deve

loped
liquid-fuel launch vehicle to orbit the Earth."""
list(tokenize(text))
```

Output:

```
: ['Founded',
 'in',
 'SpaceX',
 's',
 'mission',
 'is',
 'to',
 'enable',
 'humans',
 'to',
 'become',
 'a',
 'spacefa',
 'ring',
 'civilization',
 'and',
 'a',
 'multi',
 'planet',
 'species',
 'by',
 'building',
 'a',
 'self',
 'sustaining',
 'city',
 'on',
 'Mars',
 'In',
 'SpaceX',
 's',
 'Falcon',
 'became',
 'the',
 'first',
 'privately',
 'deve',
 'loped',
 'liquid',
```

PRACTICAL 5

Aim: Illustrate part of speech tagging.

a. Part of speech Tagging and chunking of user defined text.

b. Named Enltity recognition of user defined text.

c. Named Entity recognition with diagram using NLTK corpus – treebank

Theory:

POS Tagging (Parts of Speech Tagging) is a process to mark up the words in text format for a particular part of a speech based on its definition and context. It is responsible for text reading in a language and assigning some specific token (Parts of Speech) to each word. It is also called grammatical tagging.

Example

Input: Everything to permit us**.**

Output: [('Everything', NN),('to', TO), ('permit', VB), ('us', PRP)]

Code:-

```
import nltk
from nltk import pos_tag
from nltk import RegexpParser
nltk.download()
text ="This is practical no 6".split()
print("After Split:",text)
nltk.download('averaged_perceptron_tagger')
# averaged_perceptron_tagger is used for tagging words with their parts of speech (POS)
tokens_tag = pos_tag(text)
print("After Token:",tokens_tag)

patterns= """mychunk:{<NN.?>*<VBD.?>*<JJ.?>*<CC>?}"""
chunker = RegexpParser(patterns)
print("After Regex:",chunker)
output = chunker.parse(tokens_tag)
print("After Chunking",output)
```

Output:

```
After Split: ['learn', 'php', 'from', 'guru99', 'and', 'make', 'study', 'easy']
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
After Token: [('learn', 'JJ'), ('php', 'NN'), ('from', 'IN'), ('guru99', 'NN'), ('and', 'CC'), ('make', 'VB'), ('st
After Regex: chunk.RegexpParser with 1 stages:
RegexpChunkParser with 1 rules:
       <ChunkRule: '<NN.?>*<VBD.?>*<JJ.?>*<CC>?'>
After Chunking (S
  (mychunk learn/JJ)
  (mychunk php/NN)
  from/IN
  (mychunk guru99/NN and/CC)
  make/VB
  (mychunk study/NN easy/JJ))
```

a. Named Entity recognition of user defined text.

Code:-

```
import nltk
import spacy
from spacy import displacy
from collections import Counter
import en_core_web_sm
nlp = en_core_web_sm.load()
ex = 'European authorities fined Google a record $5.1 billion on Wednesday for abusing
its
power in the mobile phone market and ordered the company to alter its practices'
ex = nlp('European authorities fined Google a record $5.1 billion on Wednesday for
abusing
its power in the mobile phone market and ordered the company to alter its practices')
print([(X.text, X.label_) for X in ex.ents])
```

Output:

```
[('European', 'NORP'), ('Google', 'ORG'), ('$5.1 billion', 'MONEY'), ('Wednesday', 'DATE')]
```

C. Named Entity recognition with diagram using NLTK corpus – treebank

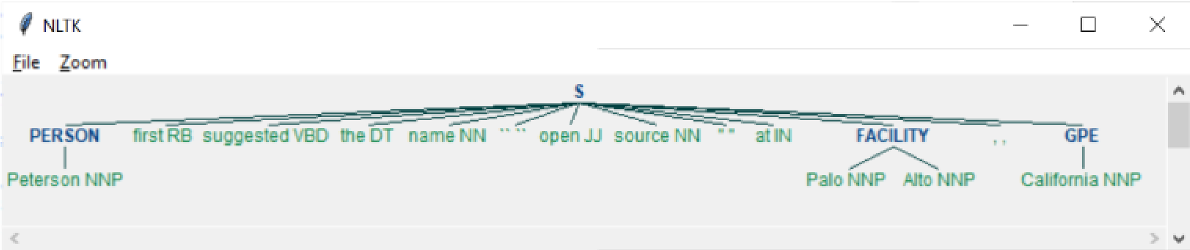
sentence = 'Peterson first suggested the name "open source" at Palo Alto, California'

```
import nltk
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
words = nltk.word_tokenize(sentence)
pos_tagged = nltk.pos_tag(words)
nltk.download('maxent_ne_chunker')
nltk.download('words')
ne_tagged = nltk.ne_chunk(pos_tagged)
print("NE tagged text:")
print(ne_tagged)
print()
print("Recognized named entities:")
for ne in ne_tagged:
if hasattr(ne, "label"):
print(ne.label(), ne[0:])
ne_tagged.draw()
```

Output:

```
NE tagged text:
(S
  (PERSON Peterson/NNP)
  first/RB
  suggested/VBD
  the/DT
  name/NN
  ``/``
  open/JJ
  source/NN
  ''/''
  at/IN
  (FACILITY Palo/NNP Alto/NNP)
  ,/,
  (GPE California/NNP))

Recognized named entities:
PERSON [('Peterson', 'NNP')]
FACILITY [('Palo', 'NNP'), ('Alto', 'NNP')]
GPE [('California', 'NNP')]
```



PRACTICAL 6

Finite state automata

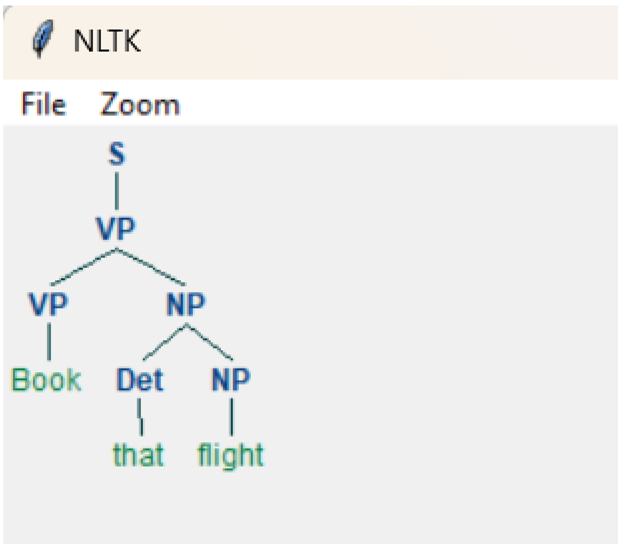
a) Define grammar using nltk. Analyze a sentence using the same.

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:

```
['Book', 'that', 'flight']
(S (VP (VP Book) (NP (Det that) (NP flight))))
```



b) Accept the input string with Regular expression of Finite Automaton: 101+.

Code:

```
def FA(s):
#if the length is less than 3 then it can't be accepted, Therefore end the process.
if len(s)<3:
return "Rejected"
#first three characters are fixed. Therefore, checking them using index
if s[0]=='1':
if s[1]=='0':
if s[2]=='1':
# After index 2 only "1" can appear. Therefore break the process if any other
character is detected
for i in range(3,len(s)):
if s[i]!='1':
return "Rejected"
return "Accepted" # if all 4 nested if true
return "Rejected" # else of 3rd if
return "Rejected" # else of 2nd if
return "Rejected" # else of 1st if
inputs=['1','10101','101','10111','01010','100','','10111101','1011111']
for i in inputs:
print(FA(i))
```

Output:

```
Rejected
Rejected
Accepted
Accepted
Rejected
Rejected
Rejected
Rejected
Accepted
```

c) Accept the input string with Regular expression of FA: (a+b)*bba.

Code:

```
def FA(s):
size=0
#scan complete string and make sure that it contains only 'a' & 'b'
for i in s:
if i=='a' or i=='b':
size+=1
else:
return "Rejected"
#After checking that it contains only 'a' & 'b'
```

```
#check it's length it should be 3 atleast
if size>=3:
#check the last 3 elements
if s[size-3]=='b':
if s[size-2]=='b':
if s[size-1]=='a':
return "Accepted" # if all 4 if true
return "Rejected" # else of 4th if
return "Rejected" # else of 3rd if
return "Rejected" # else of 2nd if
return "Rejected" # else of 1st if
inputs=['bba', 'ababbba', 'abba','abb', 'baba','bbb','']
for i in inputs:
print(FA(i))
```

Output:

```
Rejected
Rejected
Accepted
Accepted
Rejected
Rejected
Rejected
Rejected
Accepted
```

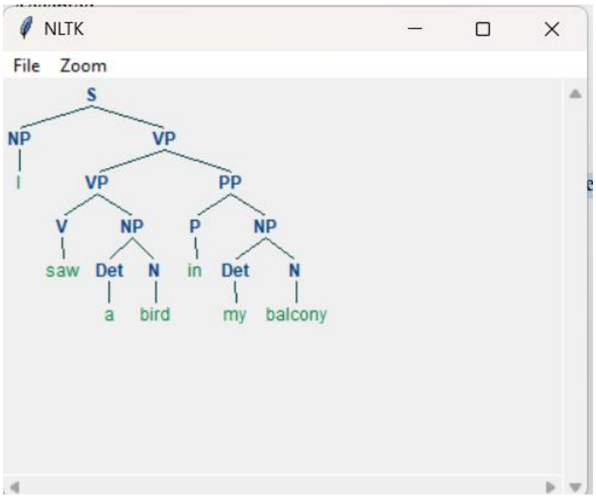
d) Implementation of Deductive Chart Parsing using context free grammar and a given sentence.

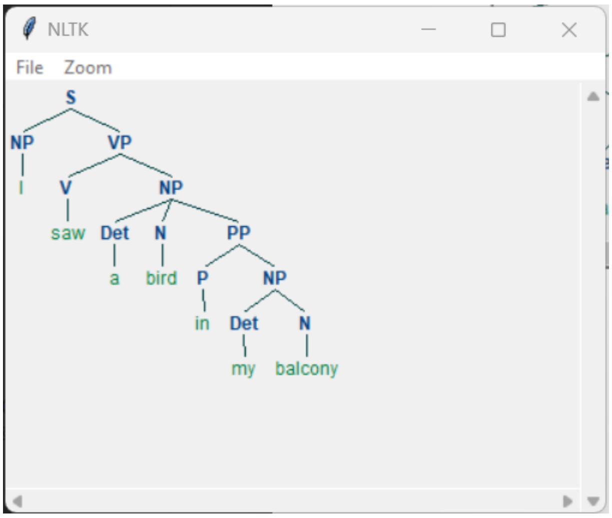
Code:-

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

```
# all_tokens = ['I', 'saw', 'a', 'bird', 'in', 'my', 'balcony']
parser = nltk.ChartParser(grammar1)
for tree in parser.parse(all_tokens):
print(tree)
tree.draw()
```

Output:





PRACTICAL 7

A : STUDY PORTER STEMMER, LANCASTER STEMMER, REGEXP STEMMER AND SNOWBALL STEMMER

a. Porter Stemmer

Code:-

```
nltk.download('punkt')
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize
# Defining the stemmer
porter = PorterStemmer()
# Taking words which have a similar stem
terms = ["gene", "genes", "genesis", "genetic", "generic", "general"]
# Performing stemming using porter stemmer on words
print("\n1. Performing porter stemming on the words")
for each_term in terms:
print(porter.stem(each_term))
# Taking a sentence
sentence = "Heya , do you know it is important to be pythonly while pythoning with
python
language. Stay being a pythoner"
# Performing stemming using porter stemmer on a sentence
print("\n2. Performing porter stemming on a sentence")
words = word_tokenize(sentence, language = 'english')
for each_word in words:
print(porter.stem(each_word))
```

Output:

```
1. Performing porter stemming on the words
gene
gene
genesi
genet
gener
gener

2. Performing porter stemming on a sentence
heya
,
do
you
know
it
is
import
to
be
pythonli
while
python
with
pythonlanguag
.
stay
be
a
python
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

b. Lancaster Stemmer

txt

(This text file is used for observing the results of stemming on a file.)

reeba - Notepad

File Edit Format View Help

Heya Reeba, A real mathematician can mathematically mathematise mathematis in a mathematical
mathematiculation. So if a mathematician can mathematise mathematics in a mathematical mathematiculation, why
can't you mathematically mathematise mathematics in a
mathematical mathematiculation.

```
import nltk
nltk.download('punkt')
from nltk.stem import LancasterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize
# Defining the stemmer
lancaster = LancasterStemmer()
# Taking words which have a similar stem
terms = ["enjoy", "enjoying", "enjoyed", "enjoyable", "enjoyment", "enjoyful"]
# Performing stemming using lancaster stemmer on words
print("\n1. Performing lancaster stemming on the words")
for each_term in terms:
print(lancaster.stem(each_term))
# Taking a sentence
sentence = "Heya, Why is it so with the dancers that when dancers dance, they dance
as if they are
dancing in the air?"
# Performing stemming using lancaster stemmer on a sentence
print("\n2. Performing lancaster stemming on a sentence")
words = word_tokenize(sentence, language = 'english')
for each_word in words:
print(lancaster.stem(each_word))
print("\n3. Performing lancaster stemming on a text file - one sentence at a time")
# Treating the text file as a collection of sentences
reeba_file = open("reeba.txt")
my_lines_list = reeba_file.readlines()
# Accessing one line at a time from the text file
words = word_tokenize(my_lines_list[0], language = 'english')
for each_word in words:
print(lancaster.stem(each_word))
```

Output:

```
dant
that
when
dant
dant
,
they
dant
as
if
they
ar
in
the
air
?

3. Performing lancaster stemming on a text file - one sentence at a time
hey
,
a
real
mathem
can
mathem
mathem
mathem
in
a
mathem
mathematicalc
.
so
if
```

c. Snowball Stemmer

Code:-

```
import nltk
nltk.download('punkt')
from nltk.stem.snowball import SnowballStemmer
# Defining the stemmer
snowball_english = SnowballStemmer("english")
snowball_english = SnowballStemmer("dutch")
# Performing stemming on one word
print("\n1. Performing snowball stemming one word")
word = snowball_english.stem("Vibing")
print(word)
# Taking a list of english words
terms = ["sid", "cheerful", "bravery","drawing", "satisfactorily", "publisher",
"painful", "hardworking",
"keys"]
# Performing stemming using snowball stemmer on words
print("\n2. Performing snowball stemming on a set of english language words")
for each_term in terms:
print(snowball_english.stem(each_term))
# Taking a list of dutch words
terms2 = ["sid", "bessen", "vriendelijkheid", "hobbelig"]
print("\n3. Performing snowball stemming on a set of dutch language words")
for each_term in terms2:
print(snowball_english.stem(each_term))
```

Output:

```
1. Performing snowball stemming one word
vibing

2. Performing snowball stemming on a set of english language words
sid
cheerful
bravery
drawing
satisfactorily
publisher
painful
hardwork
key

3. Performing snowball stemming on a set of dutch language words
sid
bess
vriendelijk
hobbel
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

d. RegExp Stemmer

Code:-

```
import nltk
nltk.download('punkt')
from nltk.stem import RegexpStemmer
# Defining the stemmer
regexp = RegexpStemmer('ing$|s$|e$|able$|ment$|less$|ly$', min=4)
# Performing stemming one word
print("\n1. Performing regexp stemming on one word at a time")
print(regexp.stem('cars'))
print(regexp.stem('bee'))
print(regexp.stem('compute'))
# Taking a list of word
terms = ["sid", "stemming", "mentally", "ease","rockstar", "frictionless",
"management","flowers",
"advisable"]
# Performing stemming using lancaster stemmer on words
print("\n2. Performing regexp stemming on a list of words")
for each_term in terms:
print(regexp.stem(each_term))
```

Output:

```
1. Performing regexp stemming on one word at a time
car
bee
comput

2. Performing regexp stemming on a list of words
sid
stemm
mental
eas
rockstar
friction
manage
flower
advis
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

PART B : STUDY WORDNET LEMMATIZER

LIBRARY :

Install the natural language toolkit library pip install nltk

```
# Program to implement WordNet Lemmatizer
import nltk
nltk.download('wordnet')
nltk.download('punkt')
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

# Initializing the Wordnet Lemmatizer
wordnet = WordNetLemmatizer()

# Performing WordNet lemmatization on single Words
print("\n1. Performing WordNet lemmatization on single Words")
print(wordnet.lemmatize("corpora"))
print(wordnet.lemmatize("best"))
print(wordnet.lemmatize("geese"))
print(wordnet.lemmatize("feet"))
print(wordnet.lemmatize("cacti"))
#Performing WordNet lemmatization on a sentence
print("\n2. Performing WordNet lemmatization on a sentence")
# Taking a sentence
sentence = "Heyaa sid, how are you doing? Keep digging in for the sentences to
observe lemmatization!"
# Tokenizing i.e. spliting the sentence into words
list_words = nltk.word_tokenize(sentence)
print("\nConverting the sentence into a list of words")
print(list_words)
# Lemmatize list of words and join
final = ' '.join([wordnet.lemmatize(each_word, pos = 'v') for each_word in list_words])
print("\nAfter applying wordnet lemmatizer, the result is....")
print(final)
```

Output:

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!

1. Performing WordNet lemmatization on single Words
corpus
best
goose
foot
cactus

2. Performing WordNet lemmatization on a sentence

Converting the sentence into a list of words
['Heyaa', 'Sid', ',', 'how', 'are', 'you', 'doing', '?', 'Keep', 'digging', 'in', 'for', 'the', 'sentences', 'to', 'observe', 'lemmatization', '

After applying wordnet lemmatizer, the result is....
Heyaa Sid , how be you do ? Keep dig in for the sentence to observe lemmatization !
```

PRACTICAL 8

Implement Naive Bayes classifier

Code:

```
import sklearn
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
breastcancer = datasets.load_breast_cancer()
print("\nFeatures of breastcancer dataset : ", breastcancer.feature_names)
print("\nLabels of breastcancer dataset : ", breastcancer.target_names)
print("\nShape of breastcancer dataset : ", breastcancer.data.shape)
print("\n--------------------------------------------------------------------------------")
R = breastcancer.data
T = breastcancer.target
# Splitting the dataset into training set and testing set
Rtrain, Rtest, Ttrain, Ttest = train_test_split(R, T, test_size = 0.2, random_state = 0)
# 1. Using the Gaussian Naive Bayes Classifier
gauss = GaussianNB()
# Training the Gaussian Naive Bayes model using training set
gauss.fit(Rtrain,Ttrain)
# Making predictions using the test set
pred = gauss.predict(Rtest)
# Generating classification report of the Gaussian Naive Bayes Model
gcr = classification_report(Ttest,pred)
print("\nClassification Report gaussian : \n", gcr)
# Generating confusion matrix of the Gaussian Naive Bayes Model
gcm = confusion_matrix(Ttest, pred)

print("\nConfusion matrix gaussian : \n", gcm)
# Evaluating the naive bayes classifier on the basis of accuracy metric
accuracy = accuracy_score(Ttest, pred)
print("\nAccuracy : ", accuracy * 100)
```

Output:

```
Features of breastcancer dataset :  ['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']

Labels of breastcancer dataset :  ['malignant' 'benign']

Shape of breastcancer dataset :  (569, 30)

----------------------------------------------------------------------------------

Classification Report gaussian :
               precision    recall  f1-score   support

           0       0.91      0.91      0.91        47
           1       0.94      0.94      0.94        67

    accuracy                           0.93       114
   macro avg       0.93      0.93      0.93       114
weighted avg       0.93      0.93      0.93       114


Confusion matrix gaussian :
 [[43  4]
 [ 4 63]]

Accuracy :  92.98245614035088
```