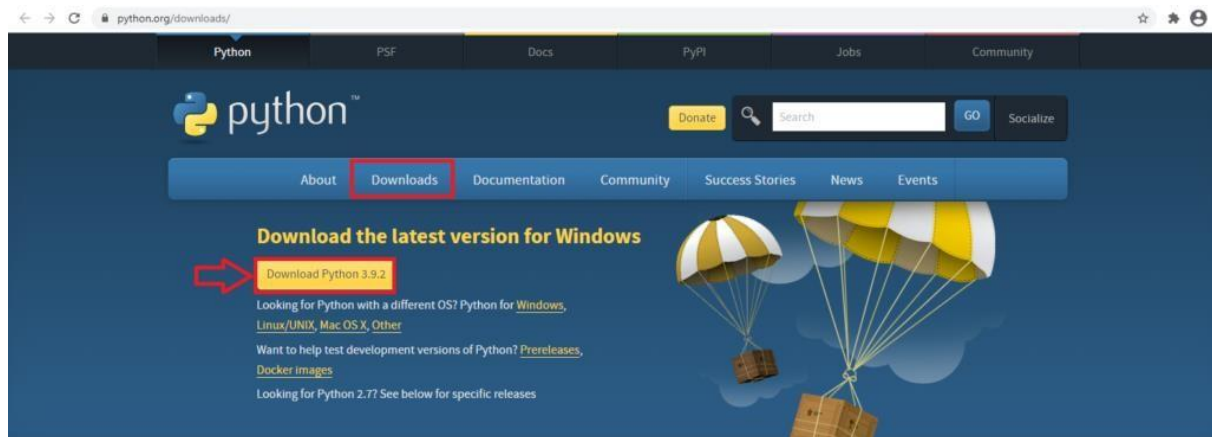


Practical 1

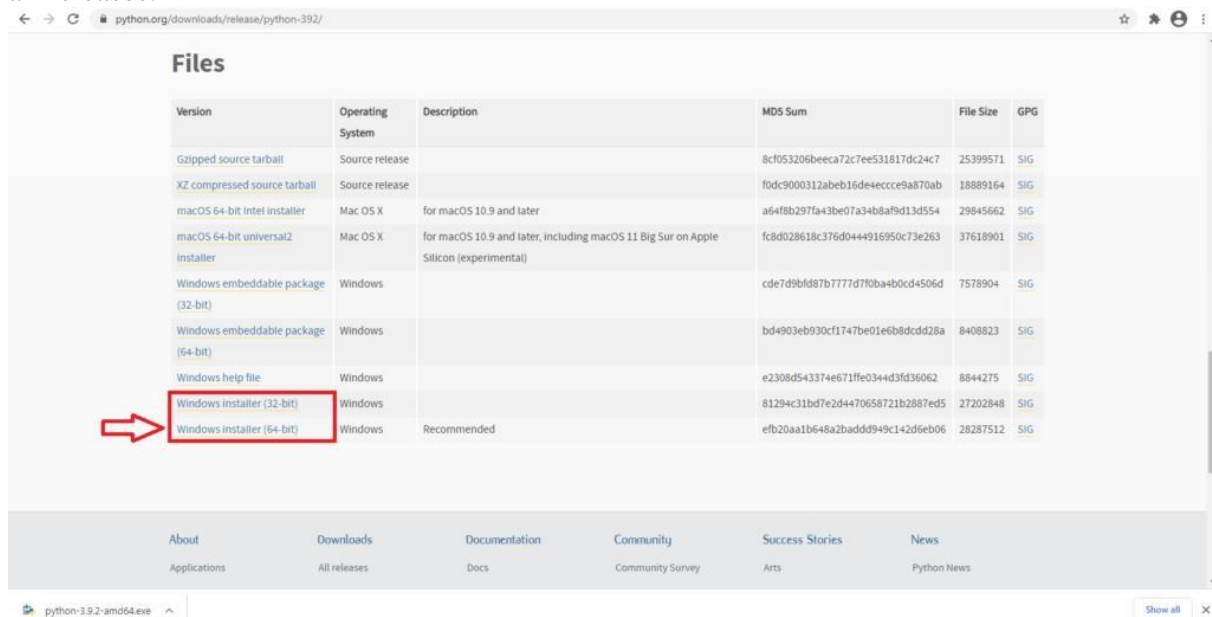
a) Install NLTK

Python 3.9.2 Installation on Windows

Step 1) Go to link <https://www.python.org/downloads/>, and select the latest version for windows.

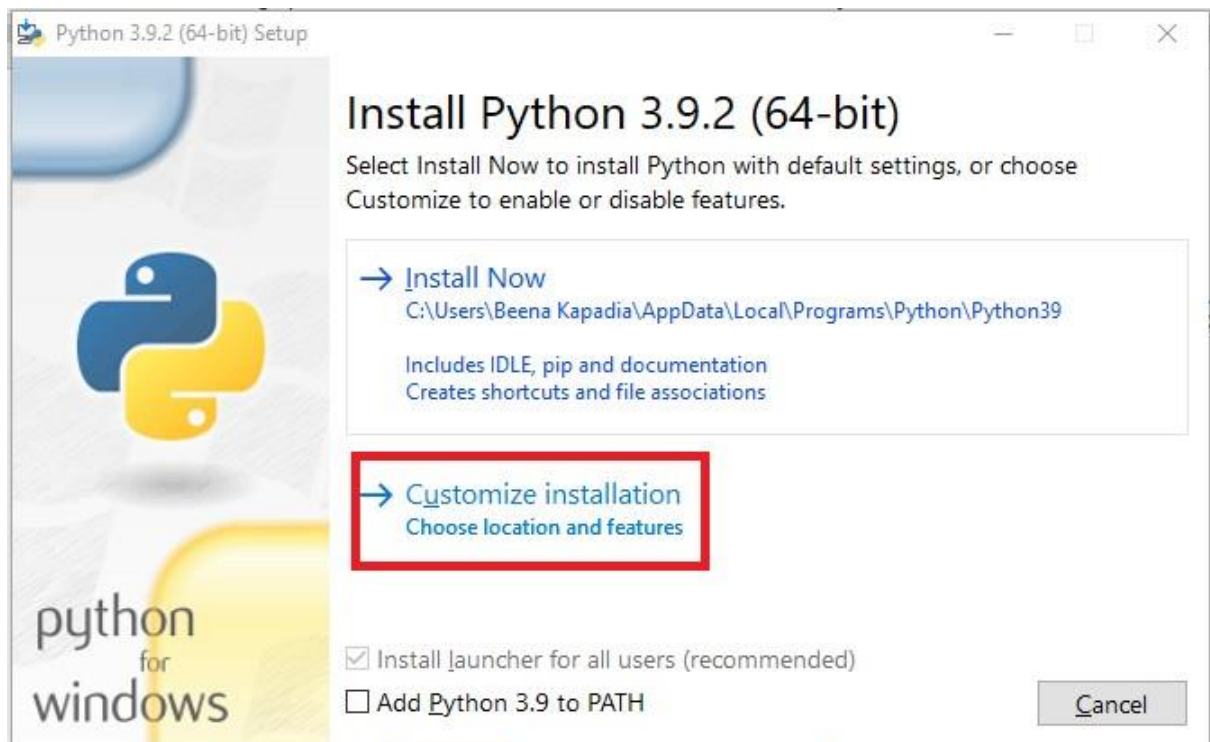


Note: If you don't want to download the latest version, you can visit the download tab and see all releases.

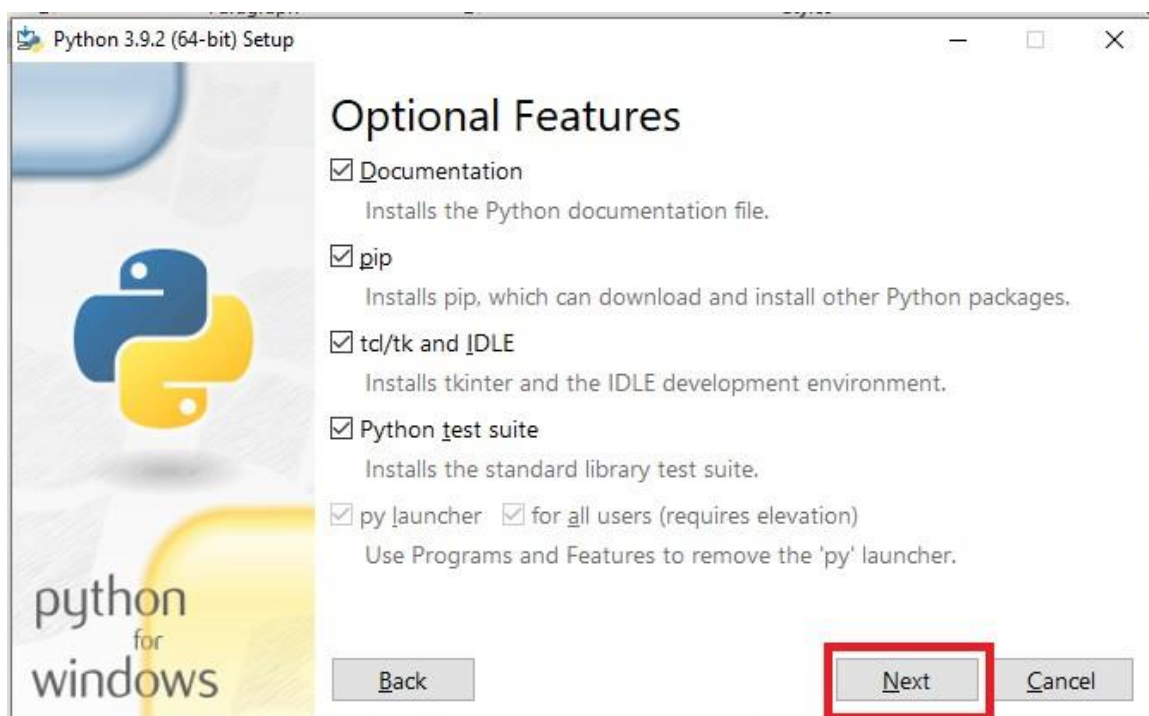


Step 2) Click on the Windows installer (64 bit)

Step 3) Select Customize Installation

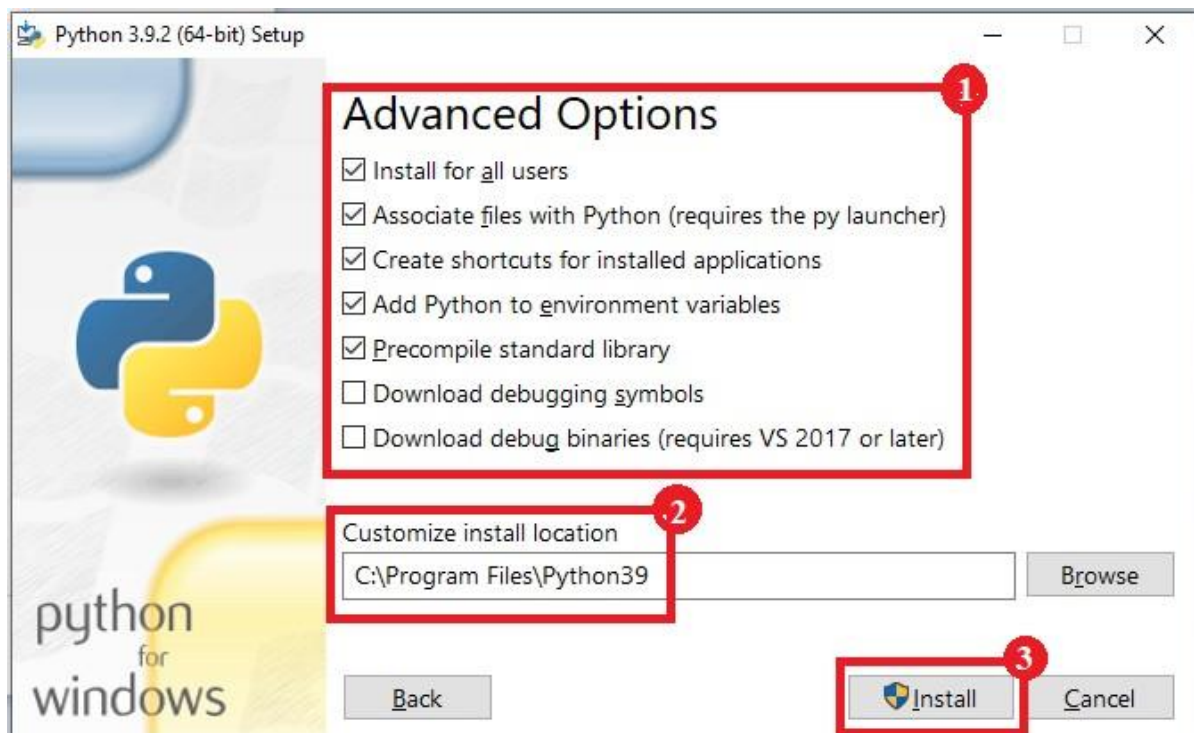


Step 4) Click NEXT



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\Beena Kapadia>pip install --upgrade pip
C:\Users\Beena Kapadia>pip install --user -U nltk
C:\Users\Beena Kapadia>>pip install --user -U numpy
C:\Users\Beena Kapadia>python
>>> import nltk
>>>
```

```
Command Prompt - python
C:\Users\Beena Kapadia>pip install --user -U nltk
Collecting nltk
  Using cached nltk-3.6.2-py3-none-any.whl (1.5 MB)
Requirement already satisfied: joblib in c:\users\beena kapadia\appdata\roaming\python\python39\site-packages (from nltk) (1.0.1)
Requirement already satisfied: tqdm in c:\users\beena kapadia\appdata\roaming\python\python39\site-packages (from nltk) (4.60.0)
Requirement already satisfied: regex in c:\users\beena kapadia\appdata\roaming\python\python39\site-packages (from nltk) (2021.4.4)
Requirement already satisfied: click in c:\users\beena kapadia\appdata\roaming\python\python39\site-packages (from nltk) (7.1.2)
Installing collected packages: nltk
  WARNING: The script nltk.exe is installed in 'C:\Users\Beena Kapadia\AppData\Roaming\Python\Python39\Scripts' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed nltk-3.6.2

C:\Users\Beena Kapadia>pip install --user -U numpy
Collecting numpy
  Using cached numpy-1.20.3-cp39-cp39-win_amd64.whl (13.7 MB)
Installing collected packages: numpy
  WARNING: The script f2py.exe is installed in 'C:\Users\Beena Kapadia\AppData\Roaming\Python\Python39\Scripts' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed numpy-1.20.3

C:\Users\Beena Kapadia>python
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import nltk
>>>
```

b. Convert the given text to speech.

Text-to-speech (TTS) is a technology that converts written text into spoken words. It is commonly used in various applications such as accessibility tools for visually impaired individuals, virtual assistants, automated customer service systems, and in-car navigation systems.

Source code:

```
# text to speech
# for google colab put ! at the start of pip.

# pip install gtts
# pip install playsound

from playsound import playsound

# import required for textto 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.

Speech-to-text technology converts spoken language from audio files into written text. It involves preprocessing the audio, extracting features, using an acoustic model to recognize speech sounds, incorporating a language model for context, and decoding to generate the transcribed text. Advanced techniques like deep learning models have significantly improved the accuracy and efficiency of speech recognition systems.

Source code:

Note: required to store the input file "male.wav" in the current folder before running the program.

```
#pip3 install SpeechRecognitionpydub
# for google colab put ! at the start of pip.

import speech_recognition as sr
filename = "male.wav"

# initialize the recognizer
```

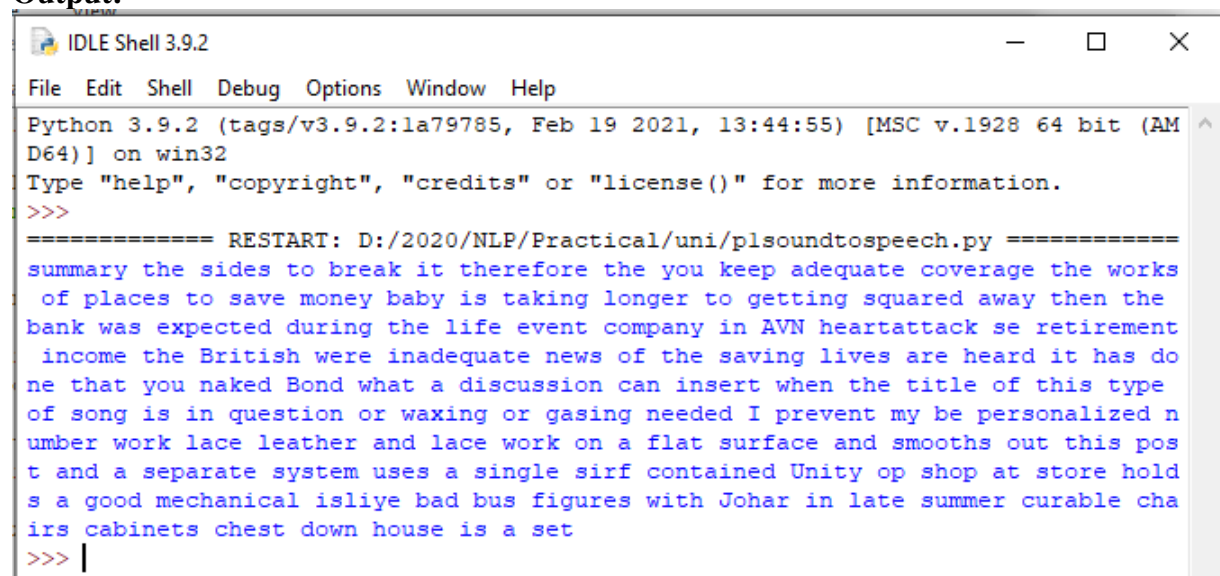
```
r = sr.Recognizer()

# 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)
```

Input:

male.wav (any wav file)

Output:



```
IDLE Shell 3.9.2
File Edit Shell Debug Options Window Help
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:/2020/NLP/Practical/uni/plsoundtospeech.py =====
summary the sides to break it therefore the you keep adequate coverage the works
  of places to save money baby is taking longer to getting squared away then the
bank was expected during the life event company in AVN heartattack se retirement
  income the British were inadequate news of the saving lives are heard it has do
ne that you naked Bond what a discussion can insert when the title of this type
of song is in question or waxing or gasing needed I prevent my be personalized n
umber work lace leather and lace work on a flat surface and smooths out this pos
t and a separate system uses a single sirf contained Unity op shop at store hold
s a good mechanical isliye bad bus figures with Johar in late summer curable cha
irs cabinets chest down house is a set
>>> |
```

Practical 2

A corpus refers to a large and structured collection of texts, spoken or written, that is used for linguistic analysis, language modeling, and machine learning tasks. It may include a wide range of text sources such as books, articles, transcripts, social media posts, and more, often organized and annotated for specific research or application purposes. Corpora play a crucial role in natural language processing (NLP) tasks like sentiment analysis, machine translation, named entity recognition, and speech recognition, providing valuable data for training and evaluating language models and algorithms.

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

source code:

```
'''NLTK includes a small selection of texts from the Project brown electronic text archive,
which contains some 25,000 free electronic books, hosted at http://www.brown.org/. We begin
by getting the Python interpreter to load the NLTK package, then ask to see
nltk.corpus.brown.fileids(), the file identifiers in this corpus:'''
```

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

```
'''Let's pick out the first of these texts — Emma by Jane Austen — and give it a short name,
emma, then find out how many words it contains:'''
ca01 = brown.words('ca01')
```

```
# display first few words
print('\nca01 has following words:\n',ca01)
```

```
# total number of words in ca01
print('\nca01 has',len(ca01),'words')
```

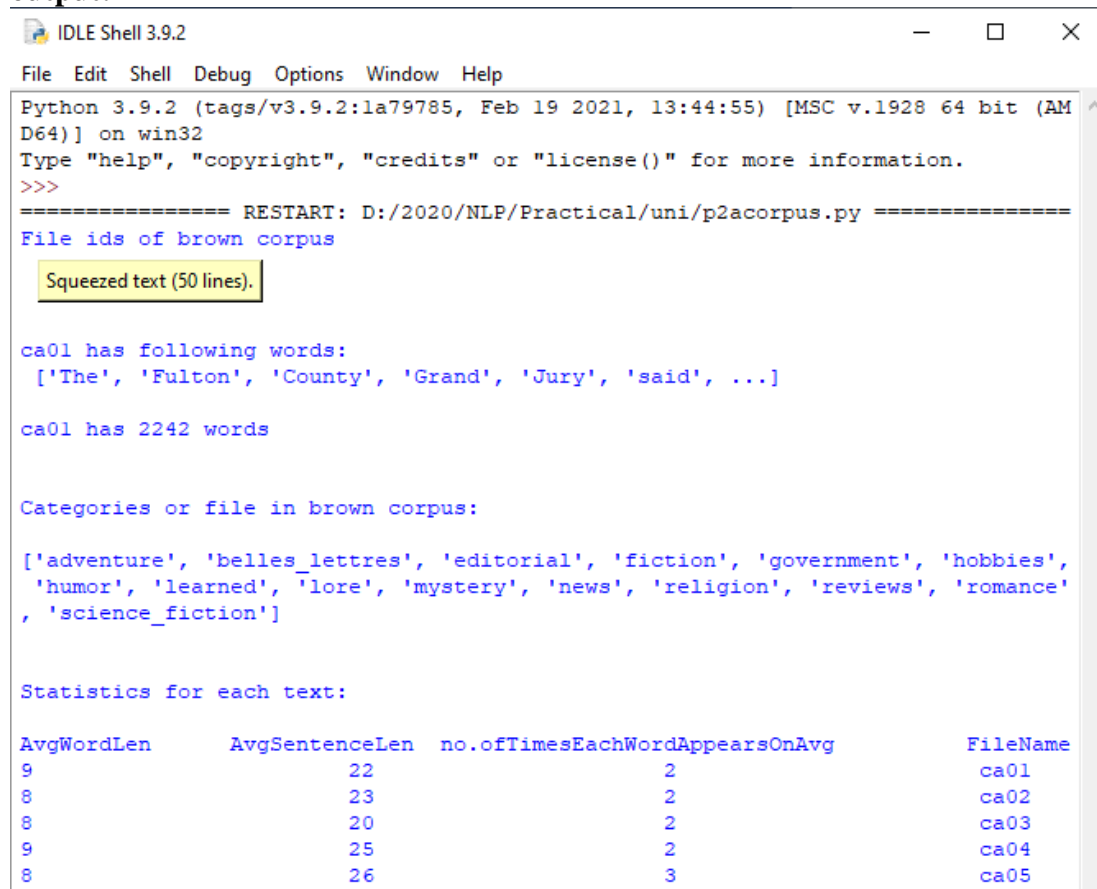
```
#categories or files
print ('\n\nCategories or file in brown corpus:\n')
print (brown.categories())
```

```
'''display other information about each text, by looping over all the values of fileid
corresponding to the brown file identifiers listed earlier and then computing statistics for each
text.'''
```

```
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:



```
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:/2020/NLP/Practical/uni/p2acorporus.py =====
File ids of brown corpus
Squeezed text (50 lines).

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']

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
```

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

source code:

"NLTK includes a small selection of texts from the Project Gutenberg electronic text archive, which contains some 25,000 free electronic books, hosted at <http://www.gutenberg.org/>. We begin by getting the Python interpreter to load the NLTK package, then ask to see `nltk.corpus.filelist.fileids()`, the file identifiers in this corpus:"

Plaintext Corpora:

```
import nltk
from nltk.corpus import PlaintextCorpusReader

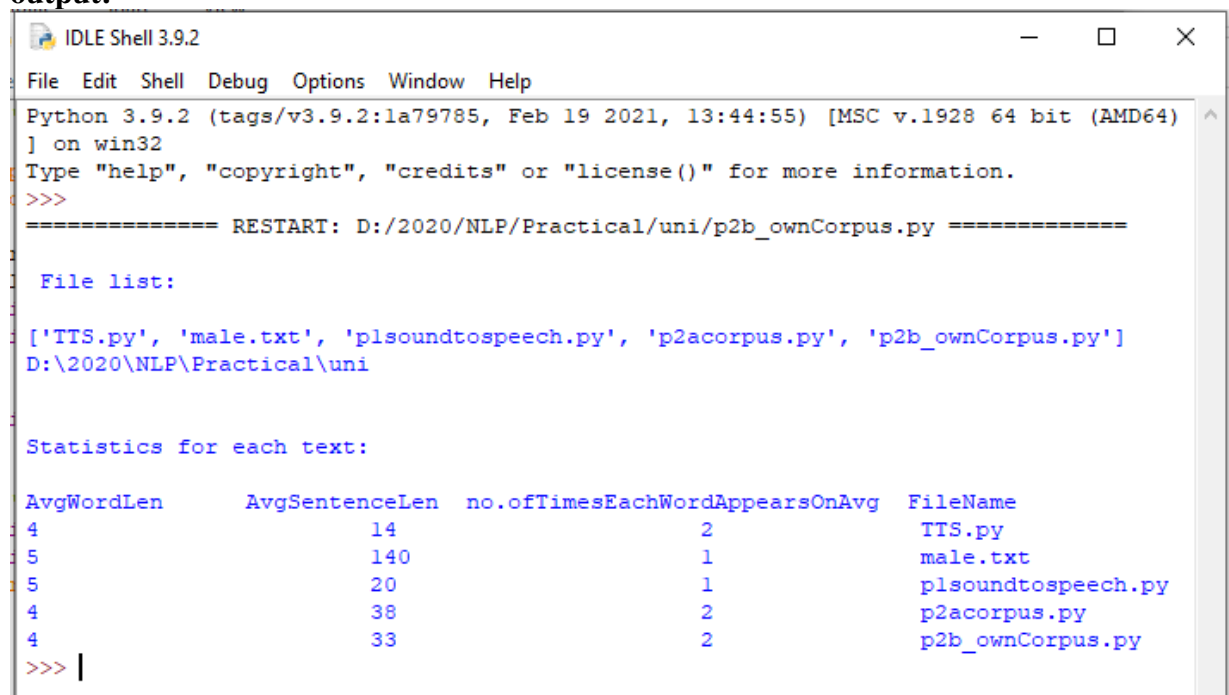
corpus_root = 'D:/2020/NLP/Practical/uni'
filelist = PlaintextCorpusReader(corpus_root, '.*')
print ('\n File list: \n')
print (filelist.fileids())

print (filelist.root)
```


"""display other information about each text, by looping over all the values of fileid corresponding to the filelist file identifiers listed earlier and then computing statistics for each text."""

```
print ('\n\nStatistics 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:



```
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:/2020/NLP/Practical/uni/p2b_ownCorpus.py =====

File list:

['TTS.py', 'male.txt', 'plsoundtospeech.py', 'p2acorp.py', 'p2b_ownCorpus.py']
D:\2020\NLP\Practical\uni

Statistics for each text:

AvgWordLen      AvgSentenceLen  no.ofTimesEachWordAppearsOnAvg  FileName
4                14              2                                TTS.py
5                140             1                                male.txt
5                20              1                                plsoundtospeech.py
4                38              2                                p2acorp.py
4                33              2                                p2b_ownCorpus.py
>>> |
```

Categorical Corpora:

```
from nltk.corpus.reader import CategorizedPlaintextCorpusReader
mycat = CategorizedPlaintextCorpusReader(
    'C:\MYCorpus_cat', r'sample.*\.txt', cat_pattern = r'.*?(one|two).*')
print ("Categorize : ", mycat.categories())
```

```
print ("\nOne : ", mycat.fileids(categories = ['one']))
```

```
print ("\nTwo : ", mycat.fileids(categories = ['two']))
```

```
mycat.words(categories='one')
```

```
print ('Avg Word Len\tAvg Sentence Len\t No of Times Each Word Appears On Avg\t
FileName')
```



```

for fileid in mycat.fileids():
    num_chars = len(mycat.raw(fileid))
    num_words = len(mycat.words(fileid))
    num_sents = len(mycat.sents(fileid))
    num_vocab = len(set([w.lower() for w in mycat.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\t\t', fileid)

```

c. Study Conditional frequency distributions

source code:

```

#process a sequence of pairs
text = ['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]
pairs = [('news', 'The'), ('news', 'Fulton'), ('news', 'County'), ...]
import nltk
from nltk.corpus import brown
fd = nltk.ConditionalFreqDist(
    (genre, word)
    for genre in brown.categories()
    for word in brown.words(categories=genre))

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

print(len(genre_word))

print(genre_word[:4])

print(genre_word[-4:])

cfd = nltk.ConditionalFreqDist(genre_word)

print(cfd)

print(cfd.conditions())

print(cfd['news'])
print(cfd['romance'])
print(list(cfd['romance']))

from nltk.corpus import inaugural
cfd = nltk.ConditionalFreqDist(
    (target, fileid[:4])
    for fileid in inaugural.fileids()
    for w in inaugural.words(fileid)
    for target in ['america', 'citizen']
    if w.lower().startswith(target))

```

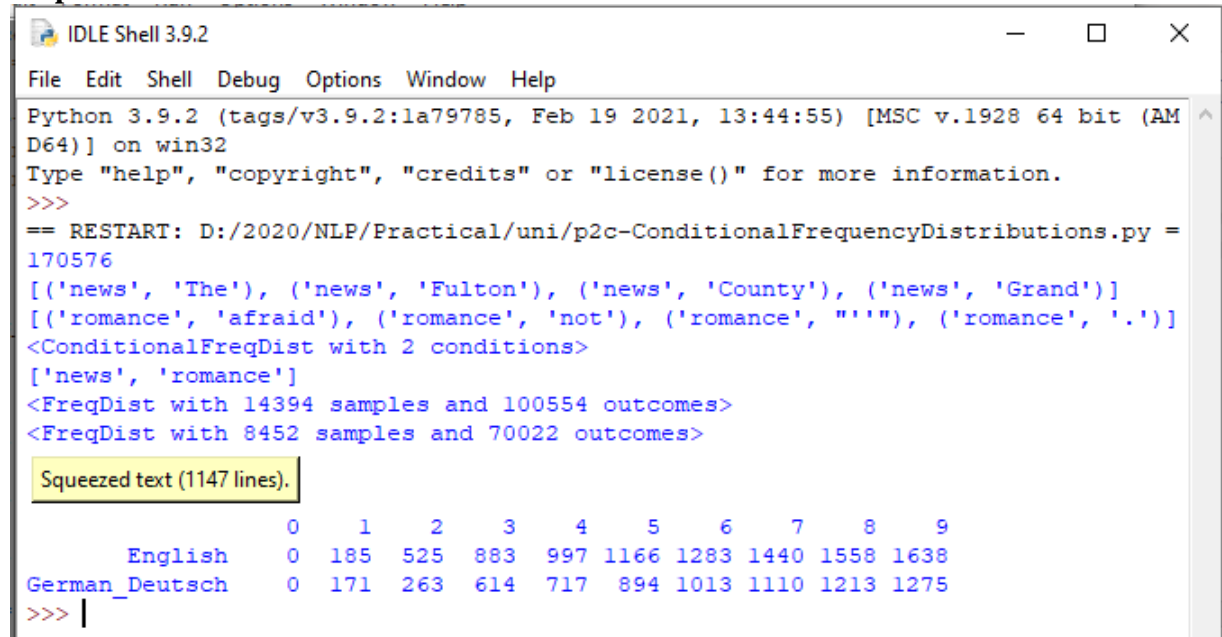
```

from nltk.corpus import udhr
languages = ['Chickasaw', 'English', 'German_Deutsch',
            'Greenlandic_Inuktitut', 'Hungarian_Magyar', 'Ibibio_Efik']
cfd = nltk.ConditionalFreqDist(
    (lang, len(word))
    for lang in languages
    for word in udhr.words(lang + '-Latin1'))

cfd.tabulate(conditions=['English', 'German_Deutsch'],
             samples=range(10), cumulative=True)

```

output:



```

Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
== RESTART: D:/2020/NLP/Practical/uni/p2c-ConditionalFrequencyDistributions.py = 170576
[('news', 'The'), ('news', 'Fulton'), ('news', 'County'), ('news', 'Grand')]
[('romance', 'afraid'), ('romance', 'not'), ('romance', ''), ('romance', '.')]
<ConditionalFreqDist with 2 conditions>
['news', 'romance']
<FreqDist with 14394 samples and 100554 outcomes>
<FreqDist with 8452 samples and 70022 outcomes>
Squeezed text (1147 lines).

```

	0	1	2	3	4	5	6	7	8	9
English	0	185	525	883	997	1166	1283	1440	1558	1638
German_Deutsch	0	171	263	614	717	894	1013	1110	1213	1275

```

>>> |

```

d. Study of tagged corpora with methods like `tagged_sents`, `tagged_words`.

A tagged corpus, also known as an annotated corpus or labeled corpus, is a type of corpus where each word or token in the text is associated with specific linguistic annotations or tags. These tags provide information about the grammatical structure, part of speech (POS), syntactic relationships, named entities, sentiment, or other linguistic features of the text.

Source code:

```

#study of tagged corpora – tagged words
nltk.corpus.brown.tagged_words()
from nltk.corpus import brown
brown_news_tagged = brown.tagged_words(categories='news', tagset='universal')
tag_fd = nltk.FreqDist(tag for (word, tag) in brown_news_tagged)
tag_fd.keys()

#Tagged Sentences
brown_tagged_sents = brown.tagged_sents(categories='news')
brown_sents = brown.sents(categories='news')
print(brown_sents)
print(brown_tagged_sents)

```

e. Write a program to find the most frequent noun tags.**Code:**

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

# checking if it is a noun or not
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)

# memoizing count
for sub in addNounWords:
    for wrd in sub.split():
        temp[wrds] += 1

# getting max frequency
res = max(temp, key=temp.get)

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

output:

```
===== RESTART: D:/2020/NLP/Practical/uni/p2emostFreq.py =====
[('Nick', 'NNP'), ('likes', 'VBZ'), ('to', 'TO'), ('play', 'VB'), ('football', 'NN'), ('.', '.'), ('Nick', 'NNP'), ('does', 'VBZ'), ('not', 'RB'), ('like', 'VB'), ('to', 'TO'), ('play', 'VB'), ('cricket', 'NN'), ('.', '.')]
['Nick', 'football', 'Nick', 'cricket']
Word with maximum frequency : Nick
>>>
```

f. Map Words to Properties Using Python Dictionaries**code:**

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

```
print(type(thisdict))
```

output:

```
===== RESTART: D:/2020/NLP/Practical/uni/p2fMap.py =====
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
Ford
3
<class 'dict'>
```

g. Study i) DefaultTagger, ii) Regular expression tagger, iii) UnigramTagger

i) DefaultTagger

code:

```
import nltk
from nltk.tag import DefaultTagger
exptagger = DefaultTagger('NN')
from nltk.corpus import treebank
testsentences = treebank.tagged_sents() [1000:]
print(exptagger.evaluate (testsentences))
```

#Tagging a list of sentences

```
import nltk
from nltk.tag import DefaultTagger
exptagger = DefaultTagger('NN')
print(exptagger.tag_sents([[ 'Hi', ',' ], [ 'How', 'are', 'you', '?' ]]))
```

output

```
===== RESTART: D:/2020/NLP/Practical/uni/p2glDefaultTagger.py =====
0.13198749536374715
[[('Hi', 'NN'), (',', 'NN')], [('How', 'NN'), ('are', 'NN'), ('you', 'NN'), ('?'
, 'NN')]]
>>> |
```

ii) Regular expression tagger,

code:

```
from nltk.corpus import brown
from nltk.tag import RegexpTagger
test_sent = brown.sents(categories='news')[0]
regexp_tagger = RegexpTagger(
    [(r'^-?[0-9]+(\.[0-9]+)?$', 'CD'), # cardinal numbers
     (r'(The|the|A|a|An|an)$', 'AT'), # articles
     (r'.*able$', 'JJ'), # adjectives
     (r'.*ness$', 'NN'), # nouns formed from adjectives
     (r'.*ly$', 'RB'), # adverbs
     (r'.*s$', 'NNS'), # plural nouns
     (r'.*ing$', 'VBG'), # gerunds
     (r'.*ed$', 'VBD'), # past tense verbs
     (r'.*', 'NN') # nouns (default)
    ])
print(regexp_tagger)
print(regexp_tagger.tag(test_sent))
```

output:

```
===== RESTART: D:/2020/NLP/Practical/uni/p2g2RegularExp.py =====
<Regex Tagger: size=9>
[('The', 'AT'), ('Fulton', 'NN'), ('County', 'NN'), ('Grand', 'NN'), ('Jury', 'N
N'), ('said', 'NN'), ('Friday', 'NN'), ('an', 'AT'), ('investigation', 'NN'), ('
of', 'NN'), ('Atlanta's', 'NNS'), ('recent', 'NN'), ('primary', 'NN'), ('electio
n', 'NN'), ('produced', 'VBD'), ('', 'NN'), ('no', 'NN'), ('evidence', 'NN'),
('', 'NN'), ('that', 'NN'), ('any', 'NN'), ('irregularities', 'NNS'), ('took',
'NN'), ('place', 'NN'), ('.', 'NN')]
```

iii) UnigramTagger**code:**

```
# Loading Libraries
from nltk.tag import UnigramTagger
from nltk.corpus import treebank

# Training using first 10 tagged sentences of the treebank corpus as data.
# Using data
train_sents = treebank.tagged_sents()[0:10]

# Initializing
tagger = UnigramTagger(train_sents)

# Lets see the first sentence
# (of the treebank corpus) as list
print(treebank.sents()[0])
print("\n", tagger.tag(treebank.sents()[0]))

# Finding the tagged results after training.
tagger.tag(treebank.sents()[0])

# Overriding the context model
tagger = UnigramTagger(model={'Pierre': 'NN'})
print("\n", tagger.tag(treebank.sents()[0]))
```

output:

```
===== RESTART: D:/2020/NLP/Practical/uni/p2g3Unigram.py =====
['Pierre', 'Vinken', ',', '61', 'years', 'old', ',', 'will', 'join', 'the', 'boa
rd', 'as', 'a', 'nonexecutive', 'director', 'Nov.', '29', '.']

[('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'), ('.', '.')]

[('Pierre', 'NN'), ('Vinken', None), (',', None), ('61', None), ('years', None)
, ('old', None), (',', None), ('will', None), ('join', None), ('the', None), ('b
oard', None), ('as', None), ('a', None), ('nonexecutive', None), ('director', No
ne), ('Nov.', None), ('29', None), ('.', None)]
```

h. 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.

Question:

Initialize the hash tag test data or URL test data and convert to plain text without any space.. Read a text file of different words and compare the plain text data with the words exist in that text file and find out different words available in that plain text. Also find out how many words

could be found. (for example, text = "#whatismyname" or text = www.whatismyname.com. Convert that to plain text without space as: whatismyname and read text file as words.txt. Now compare plain text with words given in a file and find the words from the plain text and the count of words which could be found)

Source code:

```
from __future__ import with_statement #with statement for reading file
import re # Regular expression

words = [] # corpus file words
testword = [] # test words
ans = [] # words matches with corpus

print("MENU")
print(".....")
print(" 1 . Hash tag segmentation ")
print(" 2 . URL segmentation ")
print("enter the input choice for performing word segmentation")
choice = int(input())

if choice == 1:
    text = "#whatismyname" # hash tag test data to segment
    print("input with HashTag",text)
    pattern=re.compile("[^\w]")
    a = pattern.sub("", text)
elif choice == 2:
    text = "www.whatismyname.com" # url test data to segment
    print("input with URL",text)
    a=re.split('\s|(<|>|\\d)[.,](?!\\d)', text)
    splitwords = ["www","com","in"] # remove the words which is containing in the list
    a=""
    for each in a:
        if each not in splitwords:
            a+=each+" "
    print(a)
else:
    print("wrong choice...try again")
    print(a)

for each in a:
    testword.append(each) #test word
test_lenth = len(testword) # lenth of the test data

# Reading the corpus
with open('words.txt', 'r') as f:
    lines = f.readlines()
    words =[(e.strip()) for e in lines]

def Seg(a,lenth):
    ans=[]
    for k in range(0,lenth+1): # this loop checks char by char in the corpus

        if a[0:k] in words:
            print(a[0:k],"-appears in the corpus")
    ans.append(a[0:k])
```

```

        break
    if ans != []:
        g = max(ans,key=len)
        return g

test_tot_itr = 0 #each iteration value
answer = [] # Store the each word contains the corpus
Score = 0 # initial value for score
N = 37 # total no of corpus
M = 0
C = 0
while test_tot_itr<test_lenth:
    ans_words = Seg(a,test_lenth)
    if ans_words != 0:
        test_itr = len(ans_words)
        answer.append(ans_words)
        a = a[test_itr:test_lenth]
    test_tot_itr += test_itr

Aft_Seg = " ".join([each for each in answer])
# print segmented words in the list
print("output")
print(" -----")
print(Aft_Seg) # print After segmentation the input

# Calculating Score
C = len(answer)
score = C * N / N # Calculate the score
print("Score",score)


```

Input:**Words.txt**

check	domains
domain	honesty
big	hour
rocks	follow
name	back
cheap	social
being	media
human	30
current	seconds
rates	earth
ought	this
to	is
go	insane
down	it
apple	time

what
is
my
name

let
us
go

Output:

```
IDLE Shell 3.9.2
File Edit Shell Debug Options Window Help
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:/2020/NLP/Practical/uni/p2hWord.py =====
MENU
-----
1 . Hash tag segmentation
2 . URL segmentation
enter the input choice for performing word segmentation
1
input with HashTag #whatismyname
whatismyname
what -appears in the corpus
is -appears in the corpus
my -appears in the corpus
name -appears in the corpus
output
-----
what is my name
Score 4.0
>>>
===== RESTART: D:/2020/NLP/Practical/uni/p2hWord.py =====
MENU
-----
1 . Hash tag segmentation
2 . URL segmentation
enter the input choice for performing word segmentation
2
input with URL www.whatismyname.com
whatismyname
what -appears in the corpus
is -appears in the corpus
my -appears in the corpus
name -appears in the corpus
output
-----
what is my name
Score 4.0
>>> |
```

Practical 3

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

WordNet is a lexical database of English words and their semantic relationships. It's organized as a network of synsets (sets of synonyms), where each synset represents a distinct concept or meaning. WordNet includes nouns, verbs, adjectives, and adverbs, with links between related synsets such as hypernyms (superordinates), hyponyms (subordinates), meronyms (part-whole relationships), and antonyms.

Source code:

"""WordNet provides synsets which is the collection of synonym words also called “lemmas”"""

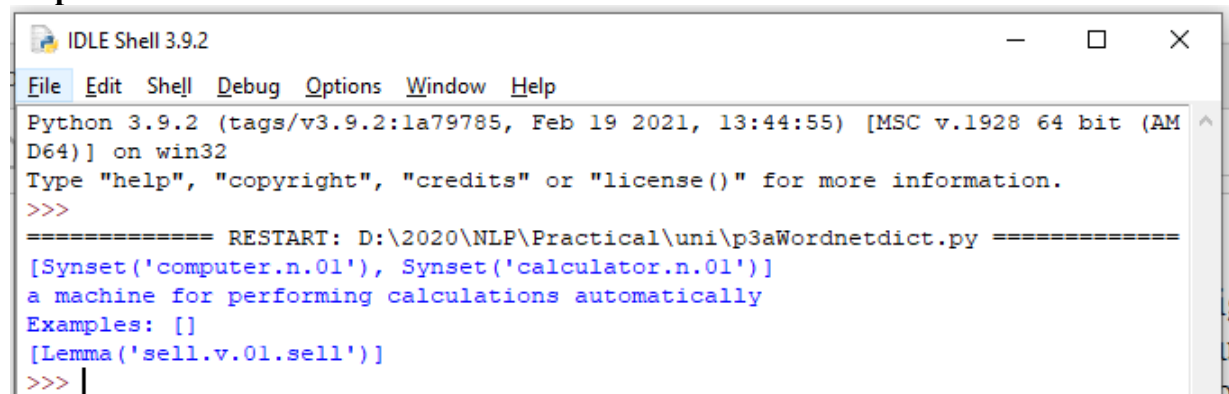
```
import nltk
from nltk.corpus import wordnet
print(wordnet.synsets("computer"))

# definition and example of the word 'computer'
print(wordnet.synset("computer.n.01").definition())

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

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

output:



```
IDLE Shell 3.9.2
File Edit Shell Debug Options Window Help
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:\2020\NLP\Practical\uni\p3aWordnetdict.py =====
[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.

A lemma refers to the base or canonical form of a word. Hyponyms are words or concepts that are more specific or subordinate to a broader category or concept, known as a hypernym. Hypernyms are words or concepts that are more general or encompassing, and they serve as the superordinate or parent categories in a semantic hierarchy.

Source 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())

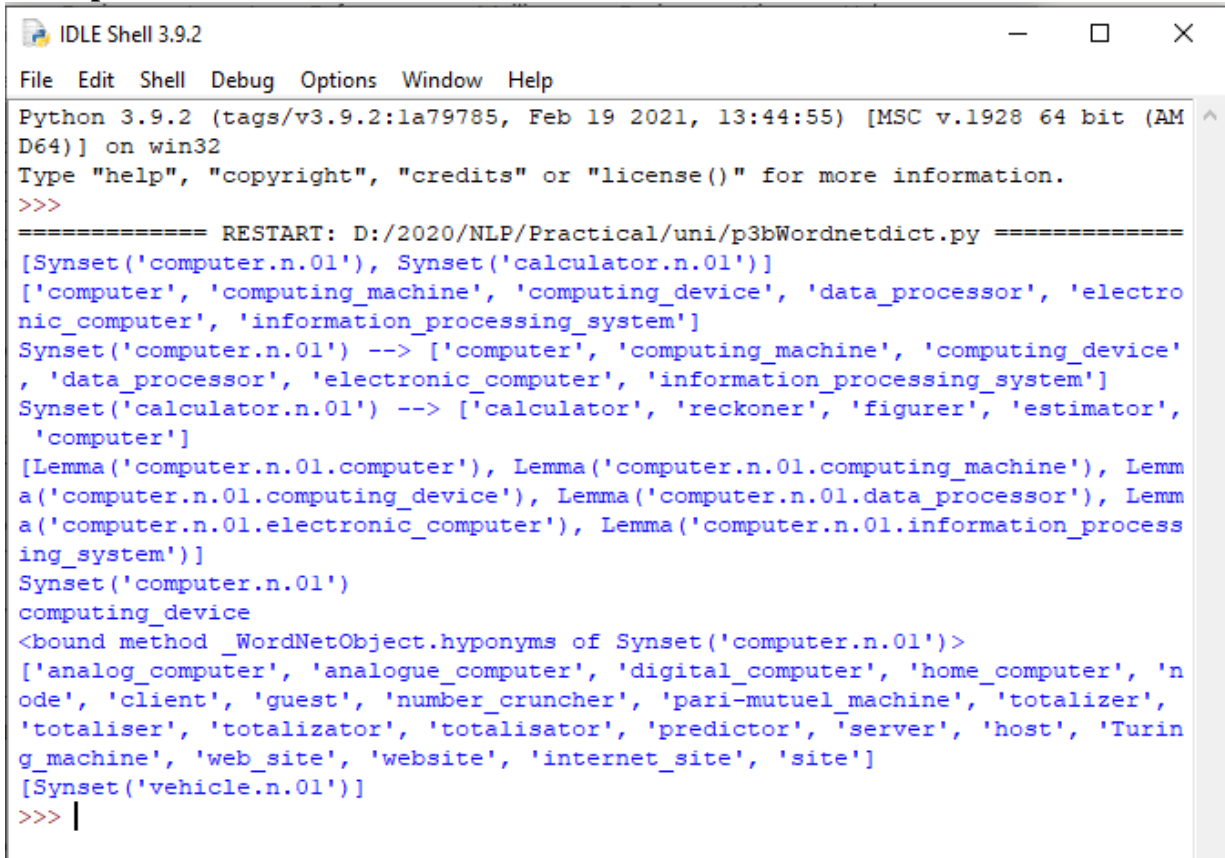
#Hyponyms give abstract concepts of the word that are much more specific
#the list of hyponyms words of the computer

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:


```

IDLE Shell 3.9.2
File Edit Shell Debug Options Window Help
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:/2020/NLP/Practical/uni/p3bWordnetdict.py =====
[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', 'information_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('computer.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.hyperonyms of Synset('computer.n.01')>
['analog_computer', 'analogue_computer', 'digital_computer', 'home_computer', 'node', 'client', 'guest', 'number_cruncher', 'pari-mutuel_machine', 'totalizer', 'totaliser', 'totalizator', 'totalisator', 'predictor', 'server', 'host', 'Turing_machine', 'web_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.

Source code:

```

from nltk.corpus import wordnet
synonyms = []
antonyms = []

for syn in wordnet.synsets("active"):
    for l in syn.lemmas():
        synonyms.append(l.name())
        if l.antonyms():
            antonyms.append(l.antonyms()[0].name())

print(set(synonyms))
print(set(antonyms))

```

d. Compare two nouns

source code:

```

import nltk
from nltk.corpus import wordnet

syn1 = wordnet.synsets('football')

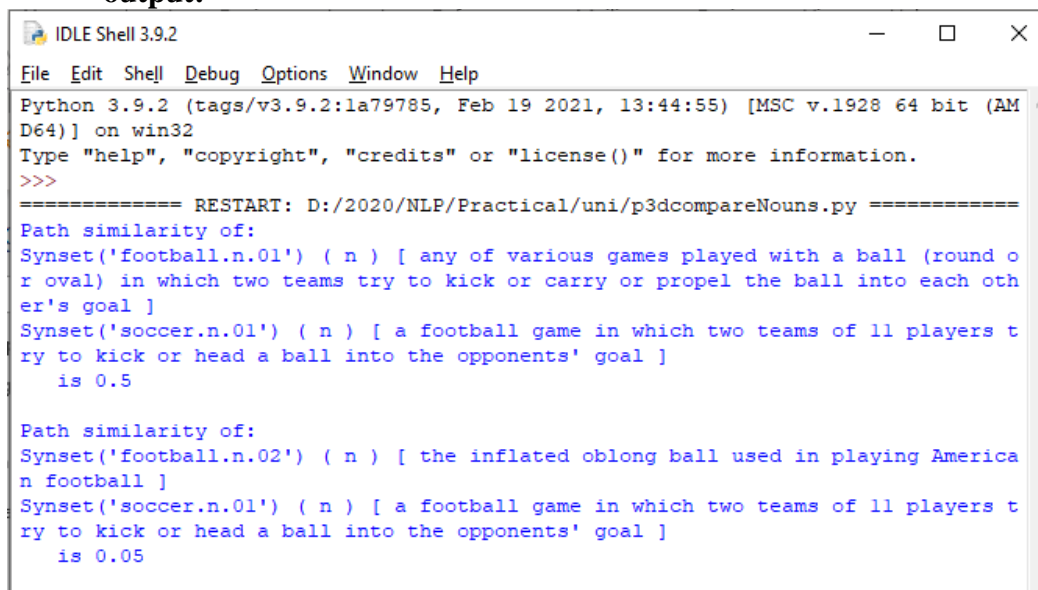
```

```
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:



```
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:/2020/NLP/Practical/uni/p3dcompareNouns.py =====
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' goal ]
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' goal ]
is 0.05
```

e. Handling stopwords:

Stopwords are commonly used words in natural language that are often filtered out during text preprocessing in natural language processing (NLP) tasks. These words are generally considered to be non-informative or redundant in the context of text analysis and can be removed to focus on more meaningful content.

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

code:

```
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)

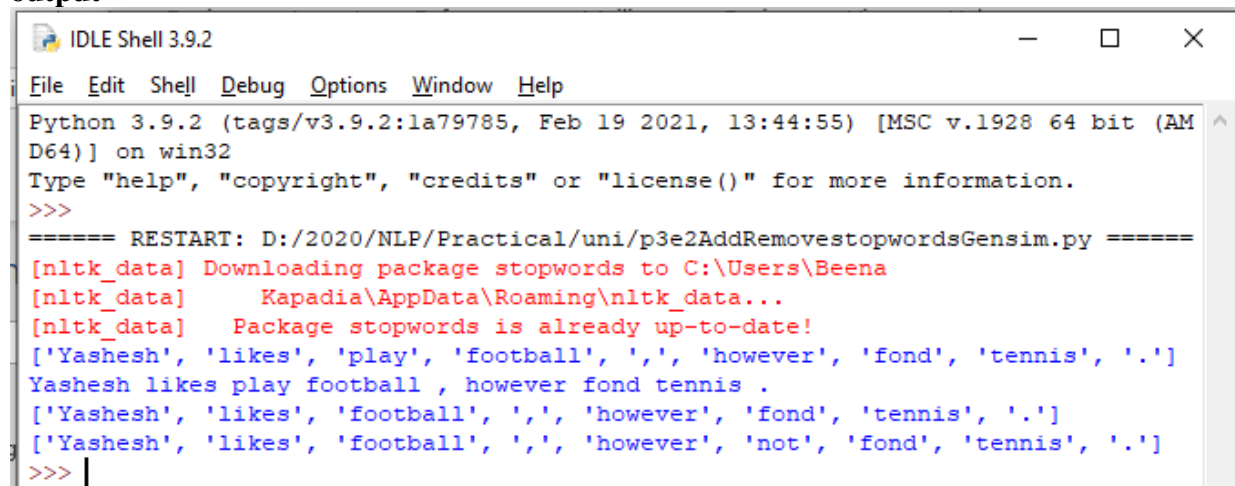
#remove 'not' from stop word collection
all_stopwords.remove('not')

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



```

IDLE Shell 3.9.2
File Edit Shell Debug Options Window Help
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:/2020/NLP/Practical/uni/p3e2AddRemovestopwordsGensim.py =====
[nltk_data] Downloading package stopwords to C:\Users\Beena
[nltk_data]   Kapadia\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
['Yashesh', 'likes', 'play', 'football', ',', 'however', 'fond', 'tennis', '.']
Yashesh likes play football , however 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

Microsoft Visual C++ 14.0 is required. Get it with "Build Tools for Visual Studio":
<https://visualstudio.microsoft.com/downloads/>

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")

```



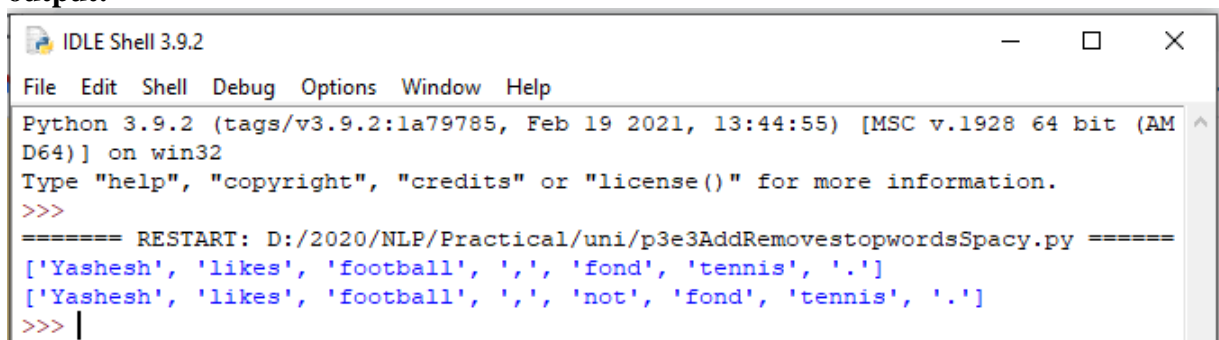
```
text = "Yashesh likes to play football, however he is not too fond of tennis."
text_tokens = word_tokenize(text)
tokens_without_sw = [word for word in text_tokens if not word in all_stopwords]

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:

```
IDLE Shell 3.9.2
File Edit Shell Debug Options Window Help
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:/2020/NLP/Practical/uni/p3e3AddRemovestopwordsSpacy.py =====
['Yashesh', 'likes', 'football', ',', 'fond', 'tennis', '.']
['Yashesh', 'likes', 'football', ',', 'not', 'fond', 'tennis', '.']
>>> |
```

Practical 4 : Text Tokenization

Tokenization is the process of breaking down a text or a corpus into smaller units called tokens. These tokens can be words, phrases, sentences, or even individual characters, depending on the level of granularity required for analysis. Tokenization is a fundamental step in natural language processing (NLP) and text analysis tasks.

a. Tokenization using Python's split() function

code:

```
text = """ This tool is an a beta stage. Alexa developers can use Get Metrics API to seamlessly
analyse metric. It also supports custom skill model, prebuilt Flash Briefing model, and the
Smart Home Skill API. You can use this tool for creation of monitors, alarms, and dashboards
that spotlight changes. The release of these three tools will enable developers to create visual
rich skills for Alexa devices with screens. Amazon describes these tools as the collection of
tech and tools for creating visually rich and interactive voice experiences. """
data = text.split('.')
for i in data:
    print (i)
```

output:

```
>>>
===== RESTART: D:/2020/NLP/Practical/uni/p4a.py =====
This tool is an a beta stage
Alexa developers can use Get Metrics API to seamlessly analyse metric
It also supports custom skill model, prebuilt Flash Briefing model, and the Sma
rt Home Skill API
You can use this tool for creation of monitors, alarms, and dashboards that spo
tlight changes
The release of these three tools will enable developers to create visual rich s
kills for Alexa devices with screens
Amazon describes these tools as the collection of tech and tools for creating v
isually rich and interactive voice experiences
```

b. Tokenization using Regular Expressions (Regex)

code:

```
import nltk
# import RegexpTokenizer() method from nltk
from nltk.tokenize import RegexpTokenizer

# Create a reference variable for Class RegexpTokenizer
tk = RegexpTokenizer('\s+', gaps = True)

# Create a string input
str = "I love to study Natural Language Processing in Python"

# Use tokenize method
tokens = tk.tokenize(str)

print(tokens)
```

output:

```
>>>
===== RESTART: D:/2020/NLP/Practical/uni/p4b.py =====
['I', 'love', 'to', 'study', 'Natural', 'Language', 'Processing', 'in', 'Python']
>>> |
```

c. Tokenization using NLTK**code:**

```
import nltk
from nltk.tokenize import word_tokenize

# Create a string input
str = "I love to study Natural Language Processing in Python"

# Use tokenize method
print(word_tokenize(str))
```

output:

```
===== RESTART: D:/2020/NLP/Practical/uni/p4c.py =====
['I', 'love', 'to', 'study', 'Natural', 'Language', 'Processing', 'in', 'Python']
>>>
```

d. Tokenization using the spaCylibrary**code:**

```
import spacy
nlp = spacy.blank("en")

# Create a string input
str = "I love to study Natural Language Processing in Python"

# Create an instance of document;
# doc object is a container for a sequence of Token objects.
doc = nlp(str)

# Read the words; Print the words
#
words = [word.text for word in doc]
print(words)
```

output:

```
===== RESTART: D:/2020/NLP/Practical/uni/p4d.py =====
['I', 'love', 'to', 'study', 'Natural', 'Language', 'Processing', 'in', 'Python']
>>>
```

e. Tokenization using Keras**code:**

```
#pip install keras
#pip install tensorflow
import keras
from keras.preprocessing.text import text_to_word_sequence

# Create a string input
str = "I love to study Natural Language Processing in Python"

# tokenizing the text
tokens = text_to_word_sequence(str)
print(tokens)
```

output:

```
>>>
===== RESTART: D:\2020\NLP\Practical\uni\p4e.py =====
['i', 'love', 'to', 'study', 'natural', 'language', 'processing', 'in', 'python']
.
```

f. Tokenization using Gensim**code:**

```
#pip install gensim

from gensim.utils import tokenize

# Create a string input
str = "I love to study Natural Language Processing in Python"

# tokenizing the text
list(tokenize(str))
```

output:

Microsoft Visual C++ 14.0 is required. Get it with "Build Tools for Visual Studio":
<https://visualstudio.microsoft.com/downloads/>

Practical 5 : Import NLP Libraries for Indian Languages and perform

Note: Execute this practical in <https://colab.research.google.com/>

a) word tokenization in Hindi

Source code:

`!pip install torch==1.3.1+cpu -f https://download.pytorch.org/whl/torch_stable.html`

```
pip install torch==1.3.1+cpu -f https://download.pytorch.org/whl/torch\_stable.html
```

```
pip install inltk
```

```
!pip install inltk
```

```
!pip install tornado==4.5.3
```

```
from inltk.inltk import setup  
setup('hi')
```

```
from inltk.inltk import tokenize
```

```
hindi_text = "प्राकृ तिकभाषासीखनाबहुतिलचस्पहै।"
```

```
# tokenize(input text, language code)  
tokenize(hindi_text, "hi")
```

output

```
['_प्राकृ तिक', '_भाषा', '_सीखना', '_बहु', '_तिलचस्प', '_है', '।']
```

b) Generate similar sentences from a given Hindi text input

Source code:

`!pip install torch==1.3.1+cpu -f https://download.pytorch.org/whl/torch_stable.html`

```
!pip install inltk
```

```
!pip install tornado==4.5.3
```

```
from inltk.inltk import setup  
setup('hi')
```

```
from inltk.inltk import get_similar_sentences
```

```
# get similar sentences to the one given in hindi  
output = get_similar_sentences('मैं आज बहुत खुश हूँ', 5, 'hi')
```

```
print(output)
```

Output:

```
['मैं आजकल बहुत खुश हूँ', 'मैं आज अत्यंत खुश हूँ', 'मैं अभी बहुत खुश हूँ',  
'मैं वित्तमान बहुत खुश हूँ', 'मैं वित्तमान बहुत खुश हूँ']
```

c) Identify the Indian language of a text**Source code:**

```
!pip install torch==1.3.1+cpu -f https://download.pytorch.org/whl/torch\_stable.html
```

```
!pip install inltk
```

```
!pip install tornado==4.5.3
```

```
from inltk.inltk import setup  
setup('gu')
```

```
from inltk.inltk import identify_language  
#Identify the Language of given text  
identify_language('બીનલિપડિયા')
```

Output:

```
Gujarati
```

Practical 6 : Illustrate part of speech tagging.

Part-of-speech (POS) tagging, also known as grammatical tagging or word-category disambiguation, is a process in natural language processing (NLP) that assigns grammatical tags to words in a text based on their role and function within a sentence. These tags represent the syntactic category of each word, such as noun, verb, adjective, adverb, pronoun, conjunction, preposition, etc.

POS Tagging, chunking and NER:

a) sentence tokenization, word tokenization, Part of speech Tagging and chunking of user defined text.

Source 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 Beena Kapadia. Today you'll be learning NLTK."
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(words)

# POS Tagging

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

# chunking

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

Output:

sentence tokenization

=====

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

word tokenization

=====

```
['Hello', '!']
```

```
['My', 'name', 'is', 'Beena', 'Kapadia', '.']
```

```
['Today', 'you', "'ll", 'be', 'learning', 'NLTK', '.']
```

POS Tagging

=====

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

b) Named Entity recognition using user defined text.**Source code:**

```
!pip install -U spacy
```

```
!python -m spacy download en_core_web_sm
```

```
import spacy
```

```
# Load English tokenizer, tagger, parser and NER
```

```
nlp = spacy.load("en_core_web_sm")
```

```
# Process whole documents
```

```
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 very 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 interview with Recode earlier "
        "this week.")
```

```
doc = nlp(text)
```

```
# Analyse syntax
```

```
print("Noun phrases:", [chunk.text for chunk in doc.noun_chunks])
```

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

Output:

Noun phrases: ['Sebastian Thrun', 'self-driving cars', 'Google', 'few people', 'the company', 'him', 'I', 'you', 'very senior CEOs', 'major American car companies', 'my hand', 'I', 'Thrun', 'an interview', 'Recode']

Verbs: ['start', 'work', 'drive', 'take', 'tell', 'shake', 'turn', 'be', 'talk', 'say']

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

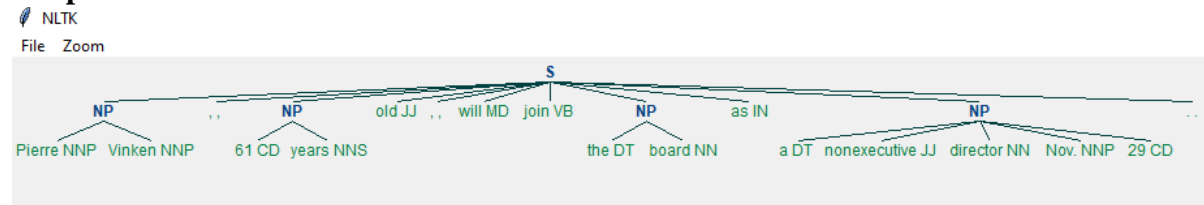
Source code:

Note: It runs on Python IDLE

```
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:



Practical 7 : Finite state automata

In NLTK (Natural Language Toolkit), grammar can be defined using context-free grammars (CFGs). A context-free grammar consists of a set of production rules that describe how symbols (such as words or phrases) in a language can be combined to form valid sentences. Each production rule specifies a symbol on the left-hand side and a sequence of symbols (including terminals and non-terminals) on the right-hand side.

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

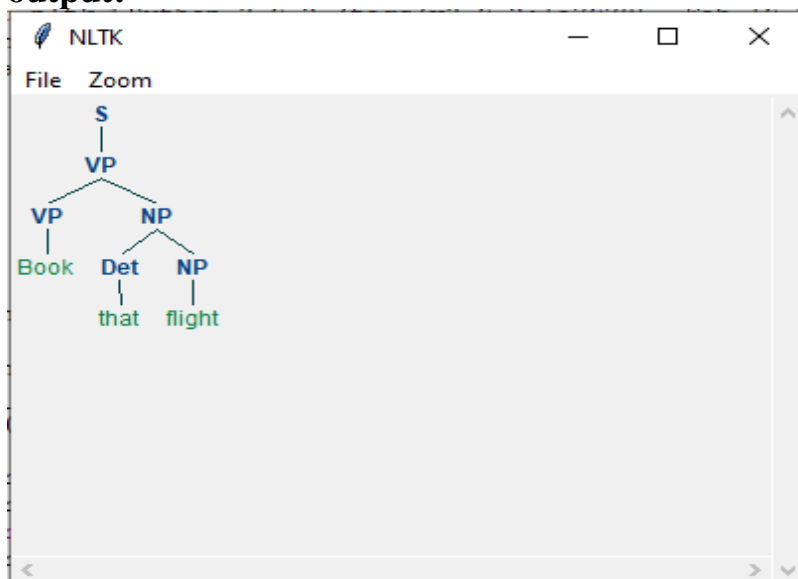
Code:

```
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:



b) Accept the input string with Regular expression of Finite Automaton: 101+.**Source 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', 'ababbbba', 'abba','abb', 'baba','bbb',""]
for i in inputs:
    print(FA(i))

```

output:

```

Rejected
Rejected
Accepted
Accepted
Rejected
Rejected
Rejected
Rejected
Rejected
Accepted

```

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

Deductive chart parsing is a technique used in natural language processing (NLP) to analyze the syntactic structure of sentences based on a context-free grammar (CFG). It involves building a parse chart that systematically explores possible parse trees according to the grammar rules, using deductive reasoning to determine valid parses.

Source 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"

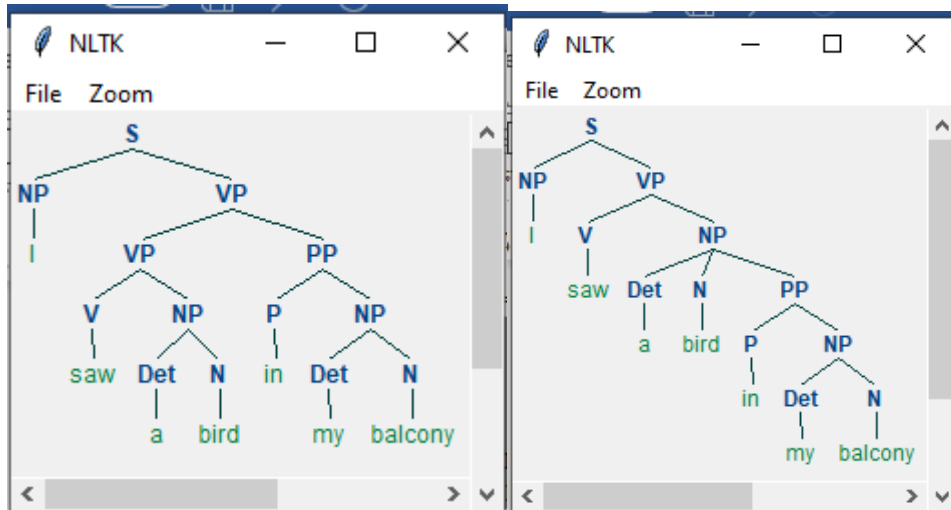
for index in range(len(sentence)):
    all_tokens = tokenize.word_tokenize(sentence)
    print(all_tokens)

# 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 8 : Study PorterStemmer, LancasterStemmer, RegexpStemmer, Snowball Stemmer, Study WordNetLemmatizer

PorterStemmer is a stemming algorithm that reduces words to their base or root form by removing common suffixes, often used in information retrieval and text mining tasks. LancasterStemmer is another stemming algorithm that is more aggressive than PorterStemmer, capable of producing shorter stems but may result in over-stemming in certain cases. RegexpStemmer is a customizable stemming algorithm in NLTK that allows users to specify their own regular expressions for stemming words based on specific patterns. Snowball Stemmer, also known as Porter2 Stemmer, is an improved version of PorterStemmer with support for multiple languages and more accurate stemming rules. WordNetLemmatizer is a lemmatization tool in NLTK that maps words to their base or dictionary forms (lemmas) based on WordNet's lexical database, useful for NLP tasks requiring accurate word normalization.

Code:

#PorterStemmer

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

Output:

```
===== RESTART: D:/2020/NLP/Practical/uni/p8aPorterStemmer.py =====
write
>>> |
```

#LancasterStemmer

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

Output:

```
===== RESTART: D:/2020/NLP/Practical/uni/p8bLancasterStemmer.py =====
write
>>> |
```

#RegexpStemmer

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

output

```
===== RESTART: D:/2020/NLP/Practical/uni/p8cRegexprStemmer.py =====
write
>>> |
```

#SnowballStemmer

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


output

```
===== RESTART: D:/2020/NLP/Practical/uni/p8dSnowballStemmer.py =====  
write  
>>> |
```

#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:

```
===== RESTART: D:/2020/NLP/Practical/uni/p8eWordNetLemmatizer.py =====  
word : lemma  
rocks : rock  
corpora : corpus  
better : good  
>>> |
```

Practical 9 : Implement Naive Bayes classifier

Naive Bayes classifier is a probabilistic machine learning model commonly used in natural language processing (NLP) for tasks such as text classification, sentiment analysis, spam filtering, and document categorization.

The Naive Bayes classifier is based on Bayes' theorem, which calculates the probability of a hypothesis given the evidence. In NLP, the hypothesis could be the class or category of a document (e.g., positive or negative sentiment), and the evidence could be the words or features in the document.

Code:

```
#pip install pandas
#pip install sklearn

import pandas as pd
import numpy as np

sms_data = pd.read_csv("spam.csv", encoding='latin-1')

import re
import nltk
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer

stemming = PorterStemmer()
corpus = []
for i in range(0, len(sms_data)):
    s1 = re.sub('[^a-zA-Z]', ' ', sms_data['v2'][i])
    s1.lower()
    s1 = s1.split()
    s1 = [stemming.stem(word) for word in s1 if word not in set(stopwords.words('english'))]
    s1 = ' '.join(s1)
    corpus.append(s1)

from sklearn.feature_extraction.text import CountVectorizer
countvectorizer = CountVectorizer()

x = countvectorizer.fit_transform(corpus).toarray()
print(x)

y = sms_data['v1'].values
print(y)

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, stratify=y, random_state=2)

#Multinomial Naïve Bayes.
from sklearn.naive_bayes import MultinomialNB
multinomialnb = MultinomialNB()
```

```
multinomialnb.fit(x_train,y_train)
```

```
# Predicting on test data:
```

```
y_pred = multinomialnb.predict(x_test)
print(y_pred)
```

```
#Results of our Models
```

```
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score
```

```
print(classification_report(y_test,y_pred))
print("accuracy_score: ",accuracy_score(y_test,y_pred))
```

input:

spam.csv file from github

output:

```
===== RESTART: D:\2020\NLP\Practical\uni\p9NaiveBayesClassifier.py =====
[[0 0 1 0 0 0 1 0 0 1 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
 1 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 1 1 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 ['ham' 'ham' 'spam' 'ham' 'ham' 'spam' 'ham' 'ham' 'spam']
['ham' 'ham' 'ham']
```

```
precision    recall  f1-score   support

   ham       0.67      1.00      0.80         2
   spam       0.00      0.00      0.00         1

 accuracy          0.67         3
  macro avg       0.33      0.50      0.40         3
 weighted avg     0.44      0.67      0.53         3

accuracy_score: 0.6666666666666666
>>> |
```

Practical 10 : Speech Tagging

Speech tagging, also known as part-of-speech tagging or POS tagging, is the process of assigning grammatical tags (parts of speech) to words in a text based on their syntactic roles and relationships within sentences. These tags indicate the word's category, such as noun, verb, adjective, adverb, pronoun, conjunction, preposition, etc.

Speech tagging is a fundamental task in natural language processing (NLP) and text analysis, as it helps in understanding the grammatical structure of sentences and extracting meaningful information from text data. It is used in various NLP applications, including information retrieval, text classification, sentiment analysis, machine translation, and named entity recognition.

i. Speech tagging using spacy

code

```
import spacy
sp = spacy.load('en_core_web_sm')
sen = sp(u"I like to play football. I hated it in my childhood though")
print(sen.text)
print(sen[7].pos_)
print(sen[7].tag_)
print(spacy.explain(sen[7].tag_))
for word in sen:
    print(f'{word.text:{12}} {word.pos_: {10}} {word.tag_: {8}} {spacy.explain(word.tag_)})')
```

```
sen = sp(u'Can you google it?')
word = sen[2]
```

```
print(f'{word.text:{12}} {word.pos_: {10}} {word.tag_: {8}} {spacy.explain(word.tag_)})')
sen = sp(u'Can you search it on google?')
word = sen[5]
```

```
print(f'{word.text:{12}} {word.pos_: {10}} {word.tag_: {8}} {spacy.explain(word.tag_)})')
```

#Finding the Number of POS Tags

```
sen = sp(u"I like to play football. I hated it in my childhood though")
```

```
num_pos = sen.count_by(spacy.attrs.POS)
num_pos
```

```
for k,v in sorted(num_pos.items()):
    print(f'{k}. {sen.vocab[k].text:{8}}: {v}')
```

#Visualizing Parts of Speech Tags

```
from spacy import displacy
```

```
sen = sp(u"I like to play football. I hated it in my childhood though")
displacy.serve(sen, style='dep', options={'distance': 120})
```

output:

```
===== RESTART: D:\2020\NLP\Practical\uni\pl0a1.py =====
I like to play football. I hated it in my childhood though
VERB
VBD
verb, past tense
I      PRON      PRP      pronoun, personal
like   VERB      VBP      verb, non-3rd person singular present
to     PART      TO       infinitival "to"
play   VERB      VB       verb, base form
football NOUN      NN       noun, singular or mass
.      PUNCT     .       punctuation mark, sentence closer
I      PRON      PRP      pronoun, personal
hated  VERB      VBD      verb, past tense
it     PRON      PRP      pronoun, personal
in     ADP       IN       conjunction, subordinating or preposition
my     PRON      PRP$     pronoun, possessive
childhood NOUN      NN       noun, singular or mass
though ADV       RB       adverb
google VERB      VB       verb, base form
google PROPON   NNP      noun, proper singular
85. ADP : 1
86. ADV : 1
92. NOUN : 2
94. PART : 1
95. PRON : 4
97. PUNCT : 1
100. VERB : 3

Using the 'dep' visualizer
Serving on http://0.0.0.0:5000 ...
```

To view the dependency tree, type the following address in your browser:
<http://127.0.0.1:5000/>. You will see the following dependency tree:

ii. Speech tagging using nltk**code:**

```
import nltk
from nltk.corpus import state_union
from nltk.tokenize import PunktSentenceTokenizer

#create our training and testing data:
train_text = state_union.raw("2005-GWBush.txt")
sample_text = state_union.raw("2006-GWBush.txt")

#train the Punkt tokenizer like:
custom_sent_tokenizer = PunktSentenceTokenizer(train_text)

# tokenize:
tokenized = custom_sent_tokenizer.tokenize(sample_text)

def process_content():
    try:
        for i in tokenized[:2]:
            words = nltk.word_tokenize(i)
```

```
tagged = nltk.pos_tag(words)
print(tagged)
```

```
except Exception as e:
    print(str(e))
```

```
process_content()
```

output:

```

IDLE Shell 3.9.2
File Edit Shell Debug Options Window Help
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:/2020/NLP/Practical/uni/pl0a2.py =====
[('PRESIDENT', 'NNP'), ('GEORGE', 'NNP'), ('W.', 'NNP'), ('BUSH', 'NNP'), ('S', 'POS'), ('ADDRESS', 'NNP'), ('BEFORE', 'IN'), ('A', 'NNP'), ('JOINT', 'NNP'), ('SESSION', 'NNP'), ('OF', 'IN'), ('THE', 'NNP'), ('CONGRESS', 'NNP'), ('ON', 'NNP'), ('THE', 'NNP'), ('STATE', 'NNP'), ('OF', 'IN'), ('THE', 'NNP'), ('UNION', 'NNP'), ('January', 'NNP'), ('31', 'CD'), ('.', 'PUNCT'), ('2006', 'CD'), ('THE', 'NNP'), ('PRESIDENT', 'NNP'), (':', 'PUNCT'), ('Thank', 'NNP'), ('you', 'PRP'), ('all', 'DT'), ('.', 'PUNCT')]
[('Mr.', 'NNP'), ('Speaker', 'NNP'), (',', 'PUNCT'), ('Vice', 'NNP'), ('President', 'NNP'), ('Cheney', 'NNP'), (',', 'PUNCT'), ('members', 'NNS'), ('of', 'IN'), ('Congress', 'NNP'), (',', 'PUNCT'), ('members', 'NNS'), ('of', 'IN'), ('the', 'DT'), ('Supreme', 'NNP'), ('Court', 'NNP'), ('and', 'CC'), ('diplomatic', 'JJ'), ('corps', 'NN'), (',', 'PUNCT'), ('distinguished', 'JJ'), ('guests', 'NNS'), (',', 'PUNCT'), ('and', 'CC'), ('fellow', 'JJ'), ('citizens', 'NNS'), (':', 'PUNCT'), ('Today', 'VB'), ('our', 'PRP$'), ('nation', 'NN'), ('lost', 'VBD'), ('a', 'DT'), ('beloved', 'VBN'), (',', 'PUNCT'), ('graceful', 'JJ'), (',', 'PUNCT'), ('courageous', 'JJ'), ('woman', 'NN'), ('who', 'WP'), ('called', 'VBD'), ('America', 'NNP'), ('to', 'TO'), ('its', 'PRP$'), ('founding', 'NN'), ('ideals', 'NNS'), ('and', 'CC'), ('carried', 'VBD'), ('on', 'IN'), ('a', 'DT'), ('noble', 'JJ'), ('dream', 'NN'), ('.', 'PUNCT')]
>>>

```

b. Statistical parsing:

Statistical parsing is a computational linguistic technique that uses probabilistic models to analyze the syntactic structure of natural language sentences. It involves training statistical models on annotated corpora, such as treebanks, to learn the probabilities of different syntactic structures and parse trees.

Treebanks are collections of sentences annotated with their syntactic parse trees, where each node represents a grammatical constituent and edges denote syntactic relationships.

i. Usage of Give and Gave in the Penn Treebank sample

Source code:

```
#probabilistic parser
#Usage of Give and Gave in the Penn Treebank sample
```

```
import nltk
import nltk.parse.viterbi
import nltk.parse.pchart
```

```

def give(t):
    return t.label() == 'VP' and len(t) > 2 and t[1].label() == 'NP'\
           and (t[2].label() == 'PP-DTV' or t[2].label() == 'NP')\
           and ('give' in t[0].leaves() or 'gave' in t[0].leaves())

def sent(t):
    return ' '.join(token for token in t.leaves() if token[0] not in '*-0')

def print_node(t, width):
    output = "%s %s: %s / %s: %s" %\
              (sent(t[0]), t[1].label(), sent(t[1]), t[2].label(), sent(t[2]))
    if len(output) > width:
        output = output[:width] + "..."
    print (output)

for tree in nltk.corpus.treebank.parsed_sents():
    for t in tree.subtrees(give):
        print_node(t, 72)

```

Output:

```

===== RESTART: D:/2020/NLP/Practical/uni/pl0b1.py =====
gave NP: the chefs / NP: a standing ovation
give NP: advertisers / NP: discounts for maintaining or increasing ad sp...
give NP: it / PP-DTV: to the politicians
gave NP: them / NP: similar help
give NP: them / NP:
give NP: only French history questions / PP-DTV: to students in a Europe...
give NP: federal judges / NP: a raise
give NP: consumers / NP: the straight scoop on the U.S. waste crisis
gave NP: Mitsui / NP: access to a high-tech medical product
give NP: Mitsubishi / NP: a window on the U.S. glass industry
give NP: much thought / PP-DTV: to the rates she was receiving , nor to ...
give NP: your Foster Savings Institution / NP: the gift of hope and free...
give NP: market operators / NP: the authority to suspend trading in futu...
gave NP: quick approval / PP-DTV: to $ 3.18 billion in supplemental appr...
give NP: the Transportation Department / NP: up to 50 days to review any...
give NP: the president / NP: such power
give NP: me / NP: the heebie-jeebies
give NP: holders / NP: the right , but not the obligation , to buy a cal...
gave NP: Mr. Thomas / NP: only a `` qualified ' rating , rather than ``...
give NP: the president / NP: line-item veto power
>>> |

```

ii. probabilistic parser

Source 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)

```

Output:

```

===== RESTART: D:/2020/NLP/Practical/uni/pl0b2.py =====
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]
Output:
(NP (JJ old) (NNS (NNS men) (CC and) (NNS women))) (p=0.000864)
>>> |

```

c. Malt parsing:**Parse a sentence and draw a tree using malt parsing.**

Note: 1) Java should be installed.

2) maltparser-1.7.2 zip file should be copied in C:\Users\Beena Kapadia\AppData\Local\Programs\Python\Python39 folder and should be extracted in the same folder.

3) engmalt.linear-1.7.mco file should be copied to C:\Users\Beena Kapadia\AppData\Local\Programs\Python\Python39 folder

Source code:

```

# copy maltparser-1.7.2(unzipped version) and engmalt.linear-1.7.mco files to C:\Users\Beena
Kapadia\AppData\Local\Programs\Python\Python39 folder
# java should be installed
# environment variables should be set - MALT_PARSER - C:\Users\Beena
Kapadia\AppData\Local\Programs\Python\Python39\maltparser-1.7.2 and MALT_MODEL -
C:\Users\Beena Kapadia\AppData\Local\Programs\Python\Python39\engmalt.linear-1.7.mco

```

```

from nltk.parse import malt
mp = malt.MaltParser('maltparser-1.7.2', 'engmalt.linear-1.7.mco')#file
t = mp.parse_one('I saw a bird from my window.'.split()).tree()
print(t)
t.draw()

```

Output:

(saw I (bird a (from (window. my))))



Practical 11

a) Multiword Expressions in NLP

A multiword expression (MWE) refers to a sequence of words that functions as a single semantic unit, often carrying a meaning that cannot be inferred from the individual words alone. MWEs are prevalent in natural language and include phrases such as "kick the bucket" (meaning to die), "take it easy" (meaning to relax), "spill the beans" (meaning to reveal a secret), and "break a leg" (meaning good luck).

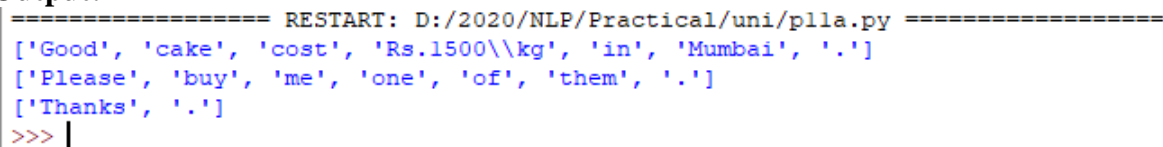
MWEs can be idiomatic, where the meaning of the expression is different from the literal meanings of its component words, or they can be compositional, where the meaning can be inferred from the individual words but the combination is still considered a single unit.

Source code:

```
# Multiword Expressions in NLP
```

```
from nltk.tokenize import MWETokenizer
from nltk import sent_tokenize, word_tokenize
s = "Good cake cost Rs.1500\kg in Mumbai. Please buy me one of them.\n\nThanks."
mwe = MWETokenizer([('New', 'York'), ('Hong', 'Kong')], separator='_')
for sent in sent_tokenize(s):
    print(mwe.tokenize(word_tokenize(sent)))
```

Output:



```
===== RESTART: D:/2020/NLP/Practical/uni/p11a.py =====
['Good', 'cake', 'cost', 'Rs.1500\\kg', 'in', 'Mumbai', '.']
['Please', 'buy', 'me', 'one', 'of', 'them', '.']
['Thanks', '.']
>>> |
```

b) Normalized Web Distance and Word Similarity

Web distance refers to measuring the semantic relatedness between words or phrases by analyzing their co-occurrence patterns in a large corpus of web data, such as search engine results or web pages. This approach leverages the vast amount of textual information available on the web to calculate similarity or distance metrics, aiding tasks like information retrieval, text classification, and sentiment analysis.

Word similarity, on the other hand, focuses on quantifying the semantic closeness between individual words based on their meaning and context, using methods like distributional semantic models, word embeddings, and semantic networks to capture semantic relationships within language.

Source code:

```
# Normalized Web Distance and Word Similarity
```

```
#convert
```

```
#Reliance supermarket
```

```
#Reliance hypermarket
```

```
#Reliance
```

```
#Reliance
```

```
#Reliance downtown
```

```
#Relianc market
#Mumbai
#Mumbai Hyper
#Mumbai dxb
#mumbai airport
#k.m trading
#KM Trading
#KM trade
#K.M. Trading
#KM.Trading
```

```
#into
```

```
#Reliance
#Reliance
#Reliance
#Reliance
#Reliance
#Reliance
#Mumbai
#Mumbai
#Mumbai
#Mumbai
#KM Trading
#KM Trading
#KM Trading
#KM Trading
#KM Trading
```

```
import numpy as np
import re
import textdistance # pip install textdistance
# we will need scikit-learn>=0.21
import sklearn #pip install sklearn
from sklearn.cluster import AgglomerativeClustering
```

```
texts = [
    'Reliance supermarket', 'Reliance hypermarket', 'Reliance', 'Reliance', 'Reliance downtown',
    'Relianc market',
    'Mumbai', 'Mumbai Hyper', 'Mumbai dxb', 'mumbai airport',
    'k.m trading', 'KM Trading', 'KM trade', 'K.M. Trading', 'KM.Trading'
]
```

```
def normalize(text):
    """ Keep only lower-cased text and numbers"""
    return re.sub('[^a-z0-9]+', '', text.lower())
```

```
def group_texts(texts, threshold=0.4):
    """ Replace each text with the representative of its cluster"""
    normalized_texts = np.array([normalize(text) for text in texts])
```

```

distances = 1 - np.array([
    [textdistance.jaro_winkler(one, another) for one in normalized_texts]
    for another in normalized_texts
])
clustering = AgglomerativeClustering(
    distance_threshold=threshold, # this parameter needs to be tuned carefully
    affinity="precomputed", linkage="complete", n_clusters=None
).fit(distances)
centers = dict()
for cluster_id in set(clustering.labels_):
    index = clustering.labels_ == cluster_id
    centrality = distances[:, index][index].sum(axis=1)
    centers[cluster_id] = normalized_texts[index][centrality.argmax()]
return [centers[i] for i in clustering.labels_]

print(group_texts(texts))

```

Output:

```

===== RESTART: D:/2020/NLP/Practical/uni/pllb.py =====
['reliance', 'reliance', 'reliance', 'reliance', 'reliance', 'reliance', 'mumbai',
 'mumbai', 'mumbai', 'mumbai', 'km trading', 'km trading', 'km trading', 'km t
rading', 'km trading']
>>> |

```

Source code:

```

#Word Sense Disambiguation
from nltk.corpus import wordnet as wn

```

```

def get_first_sense(word, pos=None):
    if pos:
        synsets = wn.synsets(word, pos)
    else:
        synsets = wn.synsets(word)
    return synsets[0]

```

```

best_synset = get_first_sense('bank')
print('%s: %s' % (best_synset.name, best_synset.definition))
best_synset = get_first_sense('set', 'n')
print('%s: %s' % (best_synset.name, best_synset.definition))
best_synset = get_first_sense('set', 'v')
print('%s: %s' % (best_synset.name, best_synset.definition))

```

Output:

```

===== RESTART: D:/2020/NLP/Practical/uni/pllc.py =====
<bound method Synset.name of Synset('bank.n.01')>: <bound method Synset.definitio
on of Synset('bank.n.01')>
<bound method Synset.name of Synset('set.n.01')>: <bound method Synset.definitio
n of Synset('set.n.01')>
<bound method Synset.name of Synset('put.v.01')>: <bound method Synset.definitio
n of Synset('put.v.01')>
>>> |

```

c. Word Sense Disambiguation

Word Sense Disambiguation (WSD) is a natural language processing task that aims to determine the correct meaning or sense of a word within a given context. Many words in natural language have multiple meanings, known as senses, and WSD algorithms are designed to identify the most appropriate sense of a word based on its usage in a specific sentence or document.

Code:-

```
import codecs
from nltk.tokenize import PunktSentenceTokenizer,sent_tokenize, word_tokenize
from nltk.corpus import stopwords, wordnet
from nltk.stem import WordNetLemmatizer, PorterStemmer
```

```
def simpleFilter(sentence):
```

```
    filtered_sent = []
    lemmatizer = WordNetLemmatizer()
    stop_words = set(stopwords.words("english"))
    words = word_tokenize(sentence)

    for w in words:
        if w not in stop_words:
            filtered_sent.append(lemmatizer.lemmatize(w))

    return filtered_sent
```

```
def simlilarityCheck(word1, word2):
```

```
    word1 = word1 + ".n.01"
    word2 = word2 + ".n.01"
    try:
        w1 = wordnet.synset(word1)
        w2 = wordnet.synset(word2)

        return w1.wup_similarity(w2)

    except:
        return 0
```

```
def synonymsCreator(word):
```

```
    synonyms = []

    for syn in wordnet.synsets(word):
        for i in syn.lemmas():
```

```

        synonyms.append(i.name())

    return synonyms

# Remove Stop Words . Word Stemming . Return new tokenised list.
def filteredSentence(sentence):

    filtered_sent = []
    lemmatizer = WordNetLemmatizer() #lemmatizes the words
    ps = PorterStemmer() #stemmer stems the root of the word.

    stop_words = set(stopwords.words("english"))
    words = word_tokenize(sentence)

    for w in words:
        if w not in stop_words:
            filtered_sent.append(lemmatizer.lemmatize(ps.stem(w)))
            for i in synonymsCreator(w):
                filtered_sent.append(i)
    return filtered_sent

if __name__ == '__main__':

    cricfile = codecs.open("E:\\Department\\MscIT_Sem_III_and sem I nov
2020\\NLP\\cricketbat.txt", 'r', "utf-8")
    sent2 = cricfile.read().lower()
    vampirefile = codecs.open("E:\\Department\\MscIT_Sem_III_and sem I nov
2020\\NLP\\vampirebat.txt", 'r', 'utf-8')
    sent1 = vampirefile.read().lower()
    sent3 = "start"

    while(sent3 != "end"):

        sent3 = input("Enter Query: ").lower()

        filtered_sent1 = []
        filtered_sent2 = []
        filtered_sent3 = []

        counter1 = 0
        counter2 = 0
        sent31_similarity = 0
        sent32_similarity = 0

        filtered_sent1 = simpleFilter(sent1)
        filtered_sent2 = simpleFilter(sent2)

```

```

filtered_sent3 = simpleFilter(sent3)

for i in filtered_sent3:

    for j in filtered_sent1:
        counter1 = counter1 + 1
        sent31_similarity = sent31_similarity + simlilarityCheck(i,j)

    for j in filtered_sent2:
        counter2 = counter2 + 1
        sent32_similarity = sent32_similarity + simlilarityCheck(i,j)

filtered_sent1 = []
filtered_sent2 = []
filtered_sent3 = []

filtered_sent1 = filteredSentence(sent1)
filtered_sent2 = filteredSentence(sent2)
filtered_sent3 = filteredSentence(sent3)

sent1_count = 0
sent2_count = 0

for i in filtered_sent3:

    for j in filtered_sent1:

        if(i==j):
            sent1_count = sent1_count + 1

    for j in filtered_sent2:
        if(i==j):
            sent2_count = sent2_count + 1

    if((sent1_count + sent31_similarity)>(sent2_count+sent32_similarity)):
        print ("Mammal Bat")
    else:
        print ("Cricket Bat")

```

Output:-

```

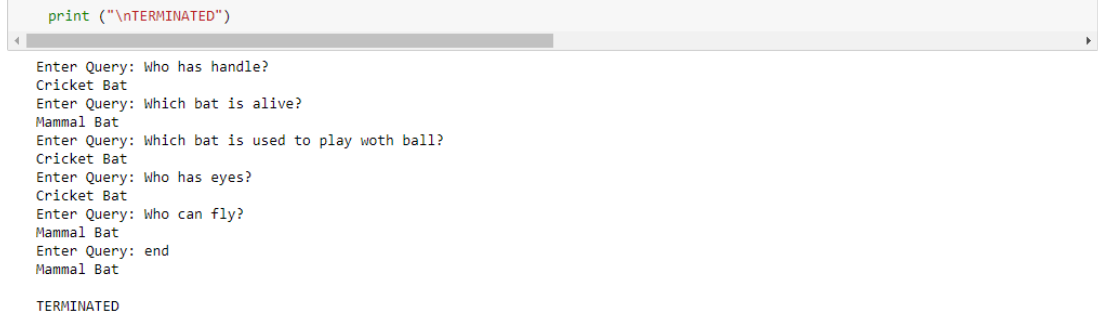
# .....
#Sentence1: the river bank has water in it and it has fishes trees . lots of water is stored in the
banks. boats float in it and animals come and drink water from it.
#sentence2: the commercial banks are used for finance. all the financial matters are managed
by financial banks and they have lots of money, user accounts like salary account and savings
account, current account. money can also be withdrawn from this bank.
#query: from which bank should i withdraw money.
#sen1: any of various nocturnal flying mammals of the order Chiroptera, having
membranous wings that extend from the forelimbs to the hind limbs or tail and anatomical
adaptations for echolocation, by which they navigate and hunt prey.

```


#sen 2: a cricket wooden bat is used for playing cricket. it is rectangular in shape and has handle and is made of wood or plastic and is used by cricket players.

print ("\nTERMINATED")

```
print ("\nTERMINATED")
```



```
Enter Query: Who has handle?
Cricket Bat
Enter Query: Which bat is alive?
Mammal Bat
Enter Query: Which bat is used to play with ball?
Cricket Bat
Enter Query: Who has eyes?
Cricket Bat
Enter Query: Who can fly?
Mammal Bat
Enter Query: end
Mammal Bat

TERMINATED
```