INDEX

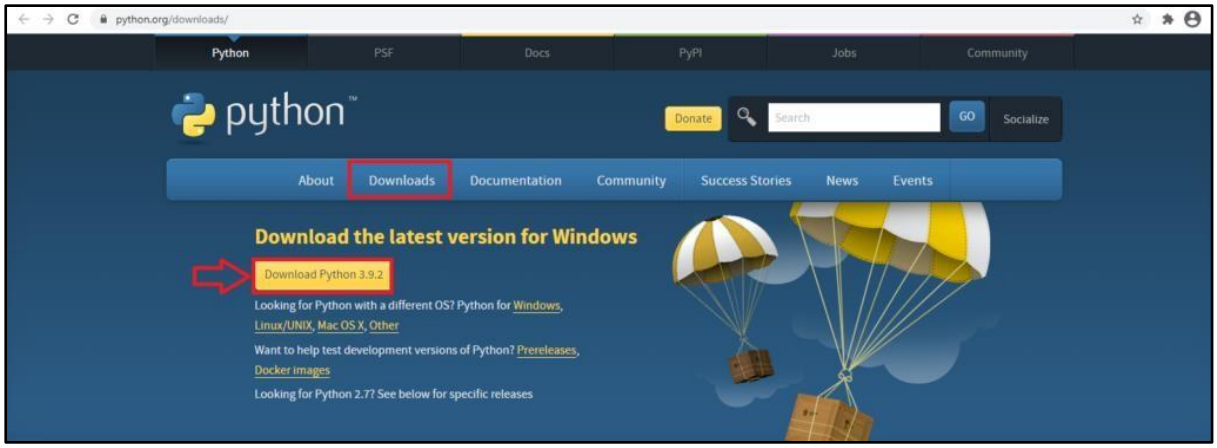
4	07/04/2023	A] Text Tokenization: Tokenization using Python's split() function. B] Tokenization using Regular Expressions (RegEx). C] Tokenization using NLTK. D] Tokenization using the spaCy library. E] Tokenization using Keras. F] Tokenization using Gensim.	39	
5	14/04/2023	A] Illustrate part of speech tagging. B] Named Entity recognition of user defined text. C] Named Entity recognition with diagram using NLTK corpus – treebank POS Tagging, chunking and NER.	45	
6	21/04/2023	A] Finite State Automata: Define grammar using nltk. Analyze a sentence using the same. B] Accept the input string with Regular expression of Finite Automaton: 101+. C] Accept the input string with Regular expression of FA: (a+b)*bba. D] Implementation of Deductive Chart Parsing using context free grammar and a given sentence.	49	
7	28/04/2023	A] Study PorterStemmer. B] Study LancasterStemmer C] Study RegexpStemmer. D] Study SnowballStemmer. E] Study WordNetLemmatizer.	57	
8	05/05/2023	Implement Naive Bayes classifier.	62	
9	12/05/2023	A] Speech Tagging. B] Statistical Parsing	64	
10	19/05/2023	A] Multiword Expressions in NLP. B] Normalized Web Distance and Word Similarity. C] Word Sense Disambiguation.	71	

PRACTICAL 1

[A] AIM: Installation of NLTK.

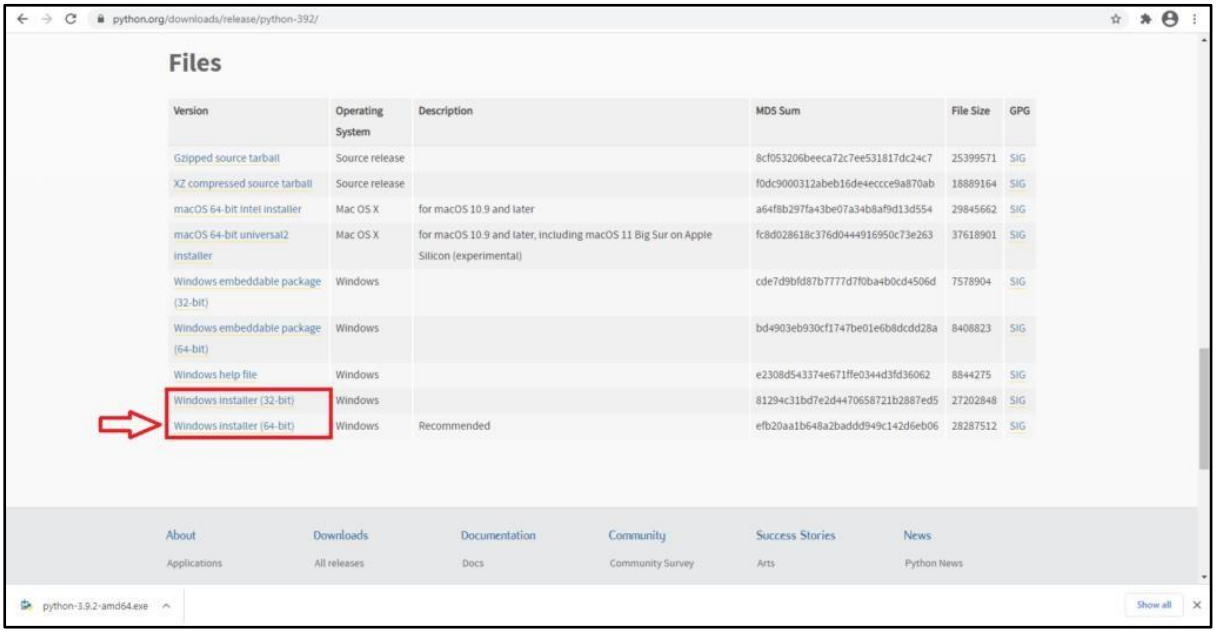
Step 1: Python Installation.

Go to link https://www.python.org/downloads/, and select the latest version for windows.

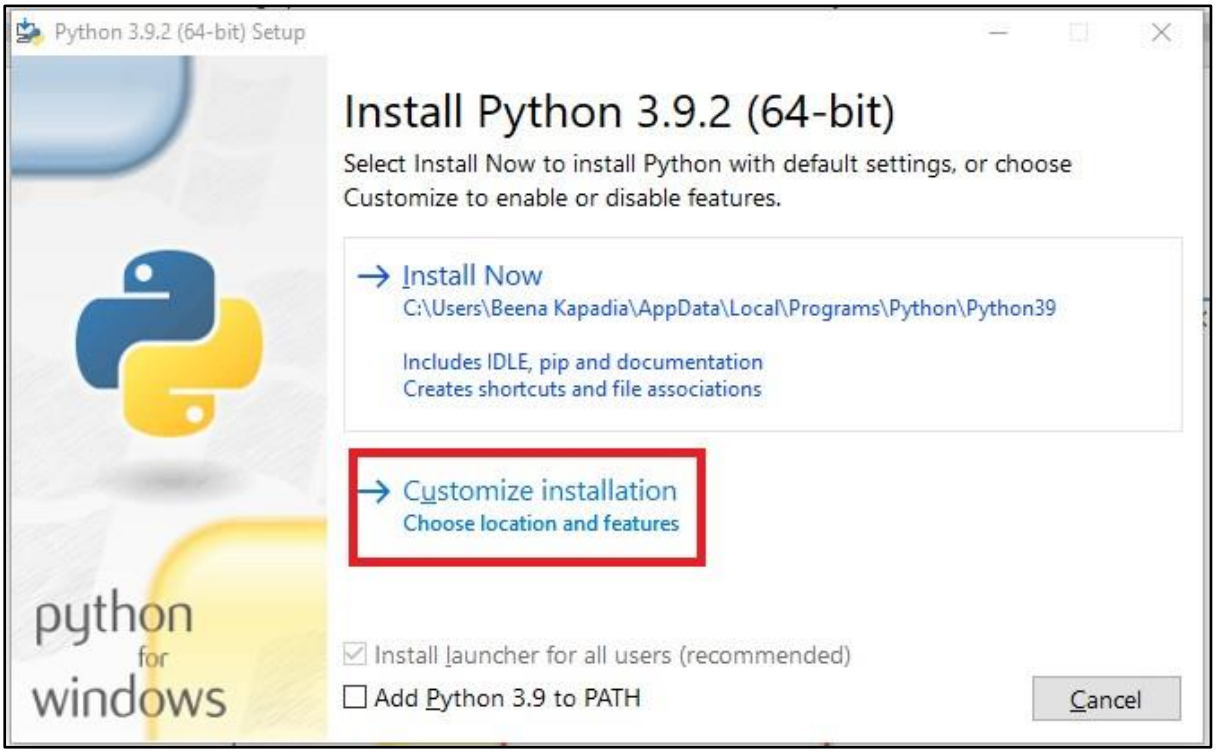


Note: If you don't want to download the latest version, you can visit the download tab and see all releases.

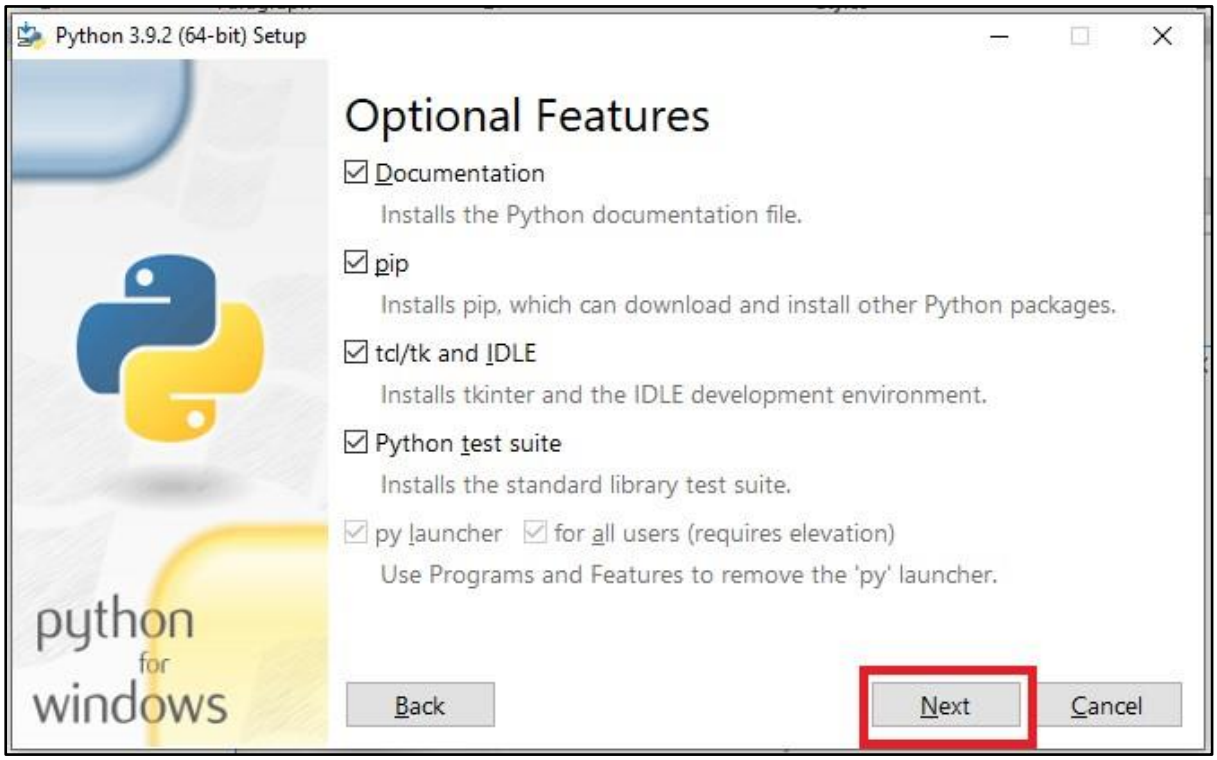
Step 2: Click on the Windows installer (64 bit).



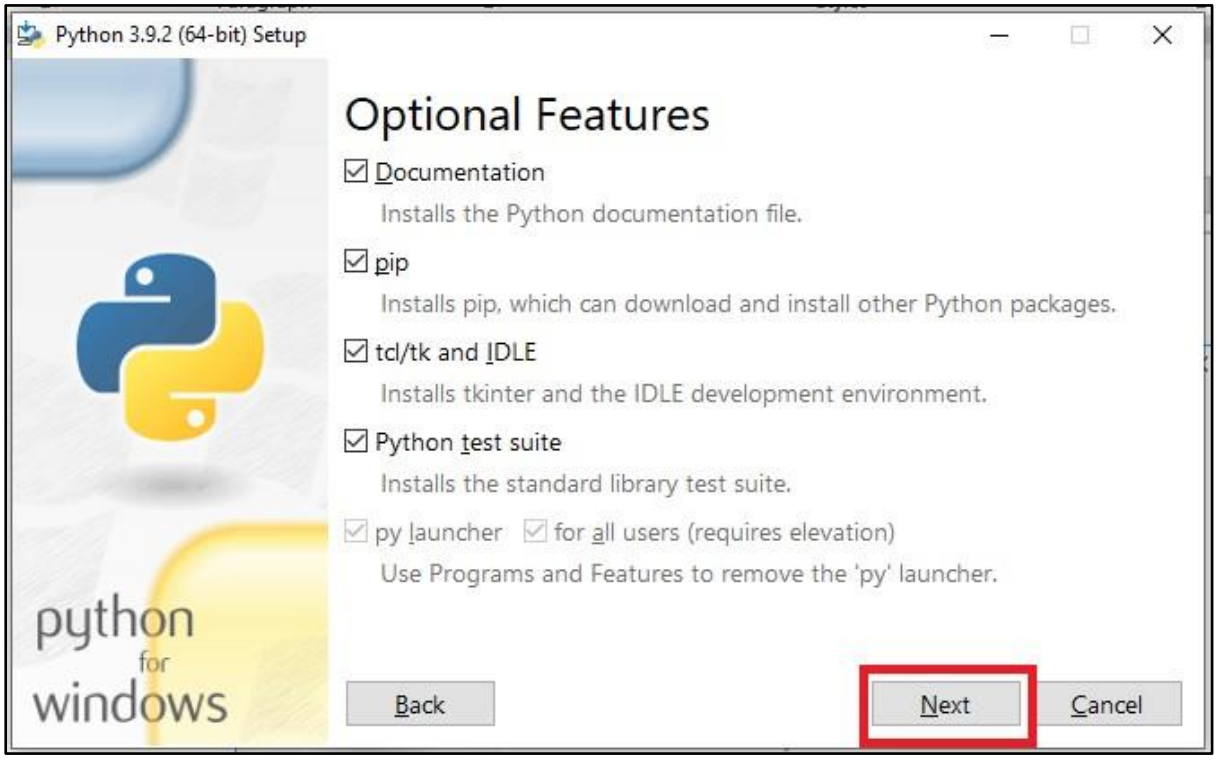
Step 3: Select Customize Installation.



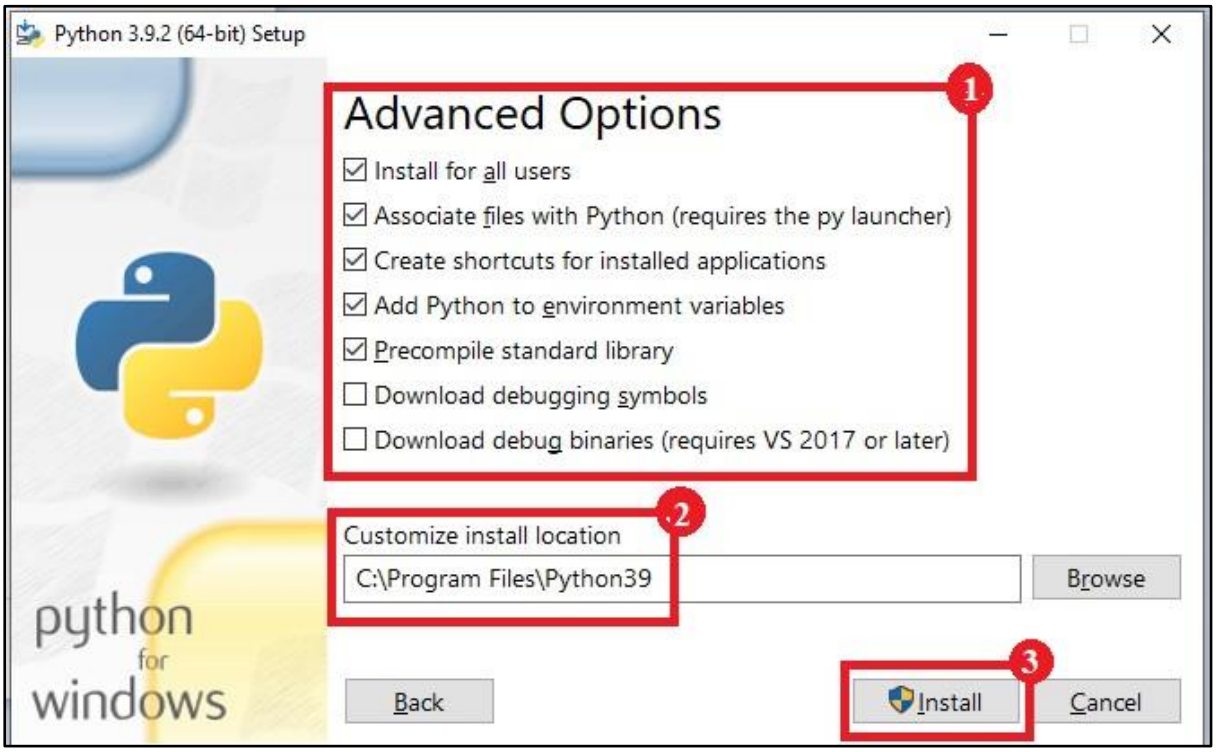
Step 4: Click "Next".



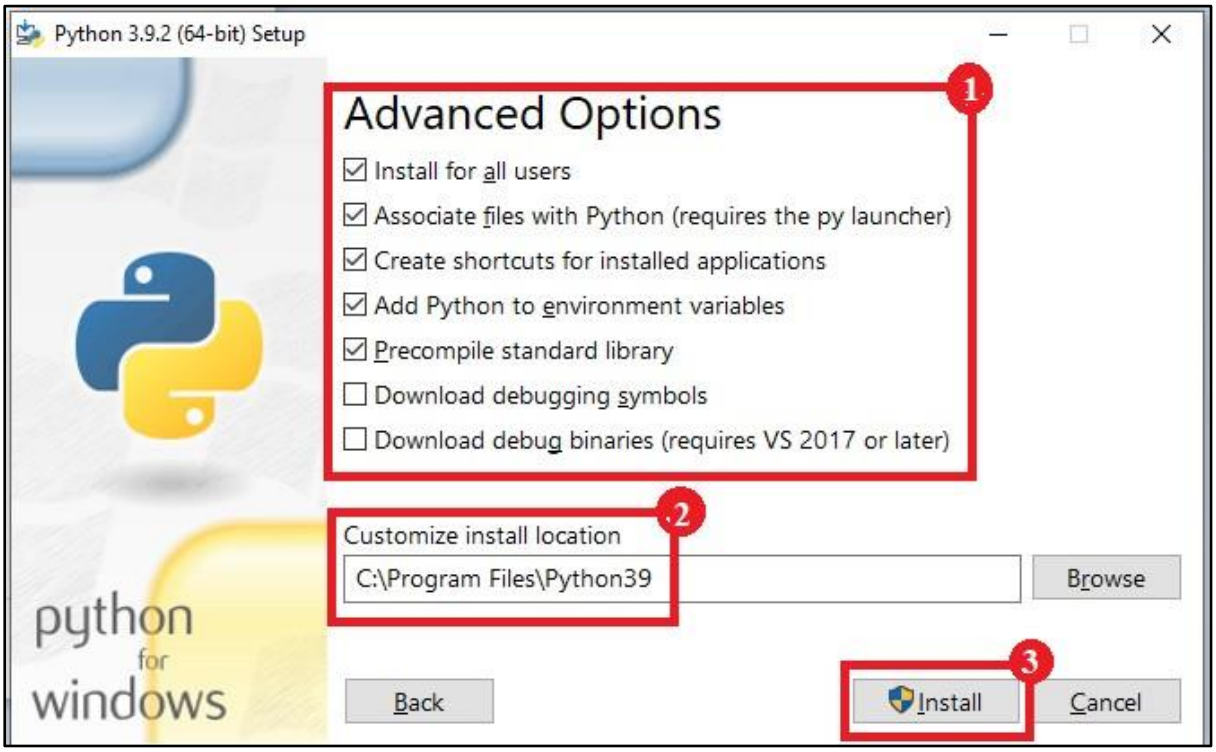
Step 5: Click "Next".



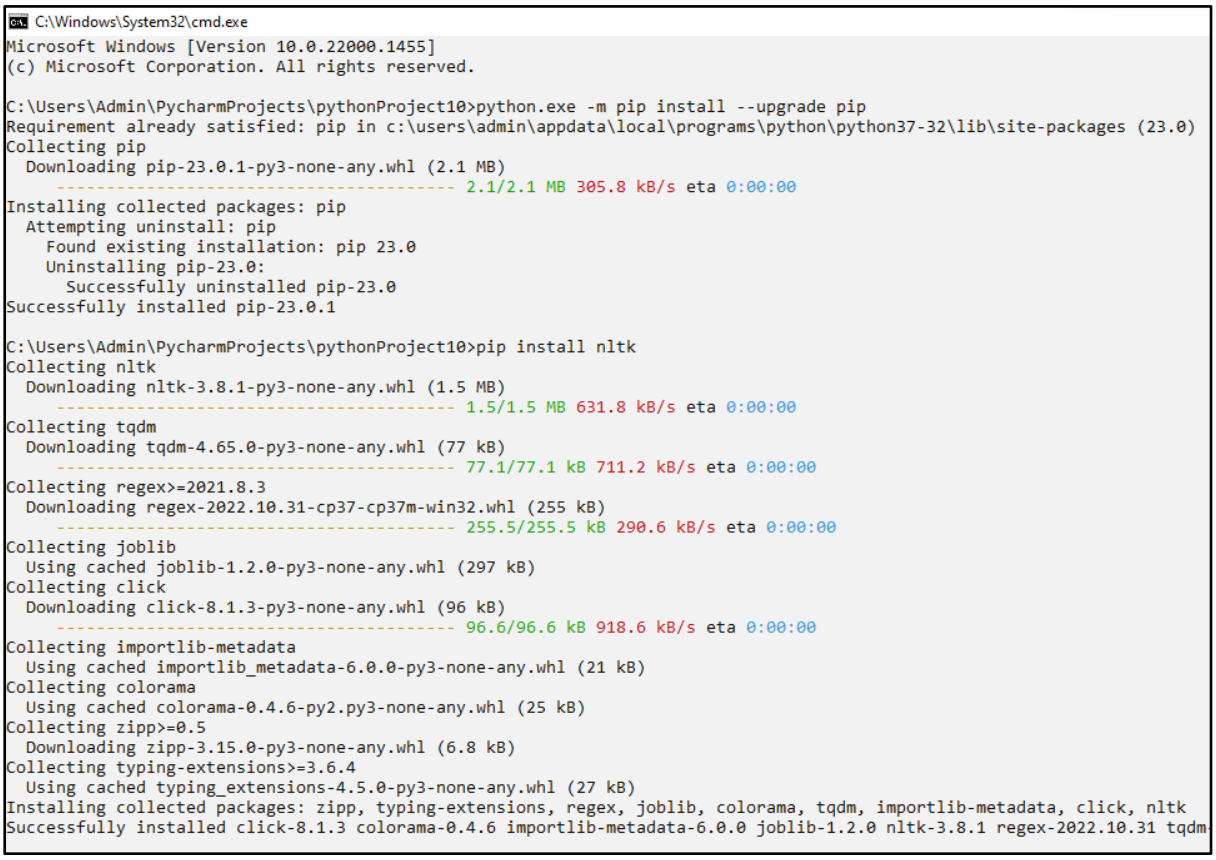
Step 6: In next screen, Select the advanced options, Give a Custom install location. Keep the default folder as c:\Program files\Python39, Click Install to complete Python Installation.



Step 7: In next screen, Select the advanced options, Give a Custom install location. Keep the default folder as c:\Program files\Python39, Click Install to complete Python Installation.



Step 8: Open command prompt window and run the following commands:
python.exe –m pip install --upgrade pip
pip install nltk



Browse https://www.nltk.org/install.html for more details

[B] AIM: Convert the given text to speech.

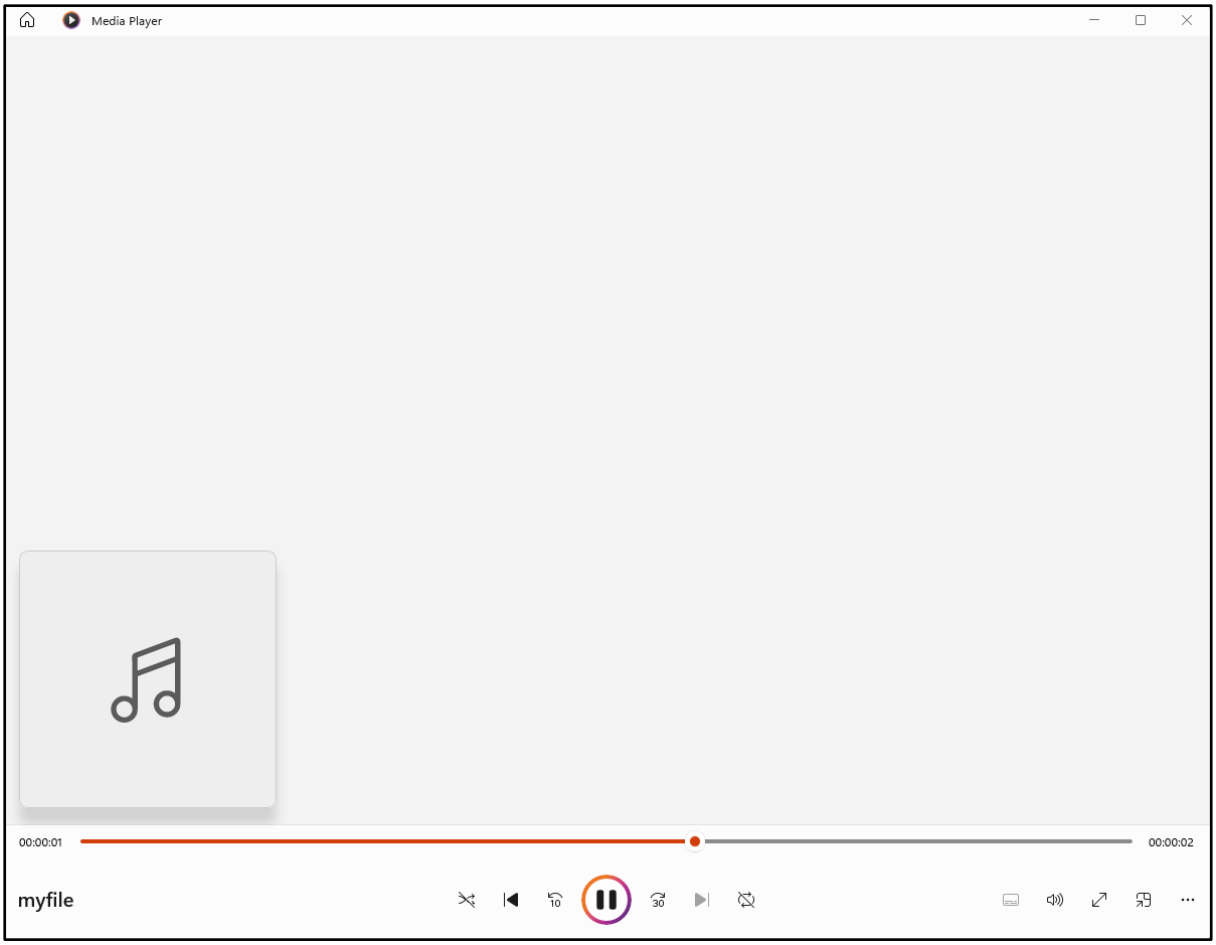
CODE:

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

OUTPUT:

```
C:\Users\Admin\PycharmProjects\NLP
myfile.mp3 has been saved.

Process finished with exit code 0
```


Media Player
00:00:01
00:00:02
myfile

[C] AIM: Convert the given speech to text.

[I] Audio file to text.

CODE:

```
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)
```

OUTPUT:

```
result2:
{   'alternative': [   {   'confidence': 0.87246531,
                           'transcript': 'what is summary decides to break it '
                                         'be careful that you keep coverage '
                                         'but look for places to save money '
                                         "maybe it's taking longer to get "
                                         'things then the bank is expected '
                                         "hiring the life for one's company "
                                         'system house is it'},
                       {   'transcript': 'what is summary decides to break it '
                                         'be careful that you keep coverage '
                                         'but look for places to save money '
                                         "maybe it's taking longer to get "
                                         'things then the bank is expected '
                                         "hiring the life for one's company "
                                         'system houses its'},
                       {   'transcript': 'what is summary decides to break it '
                                         'be careful that you keep coverage '
                                         'but look for places to save money '
                                         "maybe it's taking longer to get "
                                         'things then the bank is expected '
                                         "hiring the life for one's company"},
                       {   'transcript': 'what is summary decides to break it '
                                         'be careful that you keep coverage '
                                         'but look for places to save money '
                                         "maybe it's taking longer to get "
                                         'things then the bank is expected '
                                         "hiring the life for one's company "
                                         'system houses'},
                       {   'transcript': 'what is summary decides to break it '
                                         'be careful that you keep coverage '
                                         'but look for places to save money '
                                         "maybe it's taking longer to get "
                                         'things then the bank is expected '
                                         "hiring the life for one's company "
                                         'system house is'}],
```

```
    'final': True}
what is summary decides to break it be careful that you keep coverage but look for
 places to save money maybe it's taking longer to get things then the bank is expected
 hiring the life for one's company system house is it

Process finished with exit code 0
```

[II] Audio input to text.

CODE:

```
import speech_recognition as sr

# initialize the recognizer
r = sr.Recognizer()

with sr.Microphone() as source:

    print("Listening...")
    r.pause_threshold = 1
    r.adjust_for_ambient_noise(source)

# listen for the data (load audio to memory)
    audio = r.listen(source)
    # recognize (convert from speech to text)
    print("Recognizing")

    request = r.recognize_google(audio, language="en-in")
    print(request)
```

OUTPUT:

```
Listening...
Recognizing
result2:
{   'alternative': [   {   'confidence': 0.88687533,
                           'transcript': 'welcome to natural language '
                                         'processing'}],
    'final': True}
welcome to natural language processing
```

PRACTICAL 2

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

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
nltk.download('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:

```
File ids of brown corpus
 ['ca01', 'ca02', 'ca03', 'ca04', 'ca05', 'ca06', 'ca07', 'ca08', 'ca09', 'ca10', 'ca11', 'ca12', 'ca13', 'ca14', 'ca15', 'ca16', 'ca17', 'ca18', 'ca19', 'ca20', 'ca21', 'ca22', 'ca23',

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:

9           22          2           ca01
8           23          2           ca02
8           20          2           ca03
9           25          2           ca04
8           26          3           ca05
8           22          2           ca06
9           18          2           ca07
8           21          2           ca08
9           19          2           ca09
8           21          2           ca10
8           22          2           ca11
8           22          2           ca12
8           20          2           ca13
```

[B] AIM: Create and use your own corpora (plaintext, categorical).

CODE:

```
'''NLTK includes a small selection of texts from the Project filelist electronic text
archive, which contains some 25,000 free electronic books, hosted at
http://www.filelist.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:'''
import nltk
nltk.download('punkt')
from nltk.corpus import PlaintextCorpusReader
corpus_root = 'test'
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', fileid)
```

OUTPUT:

```
 File list:

['test.txt']
D:\sample


Statistics for each text:

4            25              2       test.txt
```

[C] AIM: Study Conditional frequency distributions.

CODE:

```
#process a sequence of pairs
text = ['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]
pairs = [('news', 'The'), ('news', 'Fulton'), ('news', 'County'), ...]
import nltk
nltk.download('brown')
nltk.download('inaugural')
nltk.download('udhr')
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_Inuktikut', '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:

```
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>
[',', '.', 'the', 'and', 'to', 'a', 'of', '``', "''", 'was', 'I', 'in', 'he', 'had', '?', 'her', 'that', 'it', 'his', 'she', 'with', 'you',
                  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] AIM: Study of tagged corpora with methods like tagged_sents, tagged_words.

CODE:

```
import nltk
from nltk import tokenize
nltk.download('punkt')
nltk.download('words')
para = "Hello! My name is UDIT. 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)
```

OUTPUT:

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

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

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

[E] AIM: Write a program to find the most frequent noun tags.

CODE:

```
import nltk
nltk.download('averaged_perceptron_tagger')
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[wrd] += 1
# getting max frequency
res = max(temp, key=temp.get)
# printing result
print("Word with maximum frequency : " + str(res))
```

OUTPUT:

```
[('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] AIM: Map Words to Properties Using Python Dictionaries.

CODE:

```
#creating and printing a dictionay 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:

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

[G] AIM: Study i) DefaultTagger, ii) Regular expression tagger, iii) UnigramTagger.

I) Default Tagger

CODE:

```
import nltk
from nltk.tag import DefaultTagger

exptagger = DefaultTagger('NN')
nltk.download('treebank')
from nltk.corpus import treebank

testsentences = treebank.tagged_sents()[1000:]
print(exptagger.accuracy(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:

```
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:

```
<Regexp Tagger: size=9>
[('The', 'AT'), ('Fulton', 'NN'), ('County', 'NN'), ('Grand', 'NN'),
 ('Jury', 'NN'), ('said', 'NN'), ('Friday', 'NN'), ('an', 'AT'),
 ('investigation', 'NN'), ('of', 'NN'), ("Atlanta's", 'NNS'), ('recent',
 'NN'), ('primary', 'NN'), ('election', 'NN'), ('produced', 'VBD'), ('``',
 'NN'), ('no', 'NN'), ('evidence', 'NN'), ("''", 'NN'), ('that', 'NN'),
 ('any', 'NN'), ('irregularities', 'NNS'), ('took', 'NN'), ('place', 'NN'),
 ('.', 'NN')]
```

III) Unigram Tagger

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()[: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:

```
['Pierre', 'Vinken', ',', '61', 'years', 'old', ',', 'will', 'join', 'the',
 'board', '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), ('board', None), ('as', None), ('a', None), ('nonexecutive',
  None), ('director', None), ('Nov.', None), ('29', None), ('.', None)]
```

[H] AIM: 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.

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 containg
in the list
    a = "".join([each for each in a if each not in splitwords])

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)
```

OUTPUT:

```
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
```

```
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] AIM: Study of Wordnet Dictionary with methods as synsets, definitions, examples, antonyms.

CODE:

```
import nltk
nltk.download('wordnet')
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:

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

[B] AIM: Study lemmas, hyponyms, hypernyms.

CODE:

```
import nltk
nltk.download('wordnet')
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', '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.hyponyms 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] AIM: Write a program using python to find synonym and antonym of word "active" using Wordnet.

CODE:

```
import nltk
nltk.download('wordnet')
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('active.s.08'),
 Synset('active.a.09'), Synset('active.a.10'), Synset('active.a.11'), Synset('active.a.12'),
 Synset('active.a.13'), Synset('active.a.14')]
[Lemma('inactive.a.02.inactive')]
```

[D] AIM: Compare two nouns.

CODE:

```
import nltk
nltk.download('wordnet')
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' 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] AIM: Handling stopwords:

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

CODE:

```
import nltk
nltk.download('stopwords')
from nltk.corpus import 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:

```
import gensim
from nltk.tokenize import word_tokenize
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:

```
Yashesh likes play football, fond tennis.
frozenset({'yourself', 'get', 'he', 'there', 'we', 'serious', 'de', 'but', 'neither', 'its',
 'after', 'among', 'did', 'part', 'itself', 'when', 'perhaps', 'less', 'again', 'co', 'none',
 'whenever', 'his', 'is', 'someone', 'yourselves', 'around', 'with', 'therein', 'while', 'one',
  'anywhere', 'up', 'various', 'via', 'rather', 'hereby', 'else', 'fire', 'the', 'alone', 'an',
  'former', 'wherein', 'us', 'due', 'many', 'everything', 'which', 'why', 'meanwhile',
 'cannot', 'first', 'since', 'fifty', 'hundred', 'anyone', 'hasnt', 'somewhere', 'about',
 'already', 'mostly', 'therefore', 'under', 'becoming', 'take', 'might', 'everywhere', 'were',
 'himself', 'twelve', 'him', 'am', 'using', 'don', 'themselves', 'her', 'would', 'herself',
 'eleven', 'full', 'becomes', 'hereafter', 'nine', 'thick', 'seems', 'last', 'once', 'was',
 'couldnt', 'for', 'nowhere', 'every', 'whither', 'not', 'had', 'it', 'will', 'sixty', 'mine',
 'become', 'fifteen', 'against', 'without', 'almost', 'five', 'make', 'above', 'side', 'off',
 'me', 'go', 'indeed', 'those', 'name', 'does', 'least', 'my', 'own', 'etc', 'whom', 'well',
 'ie', 'myself', 'they', 'very', 'six', 'un', 'now', 'whether', 'whole', 'thereupon', 'next',
 'front', 'besides', 'really', 'sometimes', 'top', 'though', 'are', 'back', 'give', 'see',
 'each', 'may', 'and', 'sincere', 'during', 'yet', 'onto', 'three', 'mill', 'thru', 'so',
 'until', 'wherever', 'namely', 'otherwise', 'ourselves', 'down', 'con', 'amoungst', 'whence',
 'something', 'on', 'out', 'hereupon', 'cant', 'them', 'then', 'their', 'anything', 'thereby',
 'here', 'never', 'anyhow', 'done', 'should', 'ever', 'than', 'your', 'a', 'more', 'latter',
 'unless', 'what', 'be', 'have', 'however', 'i', 'enough', 'do', 'became', 'km', 'ours', 'all',
```

```
  'this', 're', 'yours', 'herein', 'just', 'along', 'such', 'used', 'whatever', 'or', 'both',
 'empty', 'where', 'hers', 'within', 'being', 'to', 'thus', 'two', 'describe', 'regarding',
 'between', 'eight', 'keep', 'another', 'she', 'whereby', 'other', 'please', 'these',
 'nevertheless', 'whereupon', 'can', 'bottom', 'made', 'eg', 'into', 'too', 'nobody',
 'because', 'behind', 'who', 'how', 'you', 'below', 'towards', 'formerly', 'per', 'found',
 'thereafter', 'must', 'computer', 'fill', 'somehow', 'whoever', 'been', 'beside', 'third',
 'upon', 'kg', 'seemed', 'interest', 'forty', 'beyond', 'moreover', 'same', 'as', 'before',
 'amongst', 'further', 'everyone', 'across', 'afterwards', 'cry', 'bill', 'most', 'whereas',
 'whereafter', 'seem', 'elsewhere', 'seeming', 'that', 'quite', 'has', 'whose', 'our', 'at',
 'could', 'show', 'some', 'of', 'over', 'even', 'detail', 'always', 'any', 'say', 'nothing',
 'thence', 'doing', 'few', 'much', 'although', 'hence', 'several', 'still', 'nor', 'together',
 'throughout', 'doesn', 'twenty', 'anyway', 'also', 'by', 'ten', 'through', 'ltd', 'inc',
 'sometime', 'in', 'only', 'latterly', 'system', 'thin', 'move', 'didn', 'put', 'if', 'toward',
  'noone', 'from', 'often', 'four', 'amount', 'except', 'find', 'beforehand', 'call', 'either',
  'others', 'no'})
['Yashesh', 'football', ',', 'fond', 'tennis', '.']
['Yashesh', 'likes', 'play', 'football', ',', 'not', 'fond', 'tennis', '.']
```

III) Using Spacy Adding and Removing Stop Words in Default Spacy Stop Words List.

CODE:

```
import spacy
import nltk
from nltk.tokenize import word_tokenize
spacy.cli.download("en_core_web_sm")
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:

```
You can now load the package via spacy.load('en_core_web_sm')
['Yashesh', 'likes', 'football', ',', 'fond', 'tennis', '.']
['Yashesh', 'likes', 'football', ',', 'not', 'fond', 'tennis', '.']
```

PRACTICAL 4

[A] AIM: Text Tokenization: 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:

```
 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
```

[B] AIM: 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:

```
['I', 'love', 'to', 'study', 'Natural', 'Language', 'Processing', 'in', 'Python']
```

[C] AIM: 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:

```
['I', 'love', 'to', 'study', 'Natural', 'Language', 'Processing', 'in', 'Python']
```

[D] AIM: Tokenization using the spaCy library.

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:

```
['I', 'love', 'to', 'study', 'Natural', 'Language', 'Processing', 'in', 'Python']
```

[E] AIM: 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:

```
['I', 'love', 'to', 'study', 'Natural', 'Language', 'Processing', 'in', 'Python']
```

[F] AIM: 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
print(list(tokenize(str)))
```

OUTPUT:

```
['I', 'love', 'to', 'study', 'Natural', 'Language', 'Processing', 'in', 'Python']
```

PRACTICAL 5

AIM: Illustrate part of speech tagging.

[A] Part of speech Tagging and chunking of user defined text.

[i] sentence tokenization

[ii]word tokenization

[iii]Part of speech Tagging and chunking of user defined text.

CODE:

```
import nltk
from nltk import tokenize
nltk.download('punkt')
from nltk import tag
from nltk import chunk
nltk.download('averaged_perceptron_tagger')
nltk.download('maxent_ne_chunker')
nltk.download('words')
para = "Hello! My name is UDIT. 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 UDIT.', "Today you'll be learning NLTK."]

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

['Hello', '!']
['My', 'name', 'is', 'UDIT', '.']
['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] AIM: Named Entity recognition of user defined text.

CODE:

```
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:

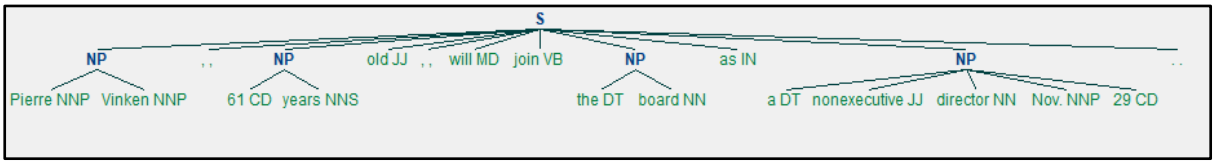
```
Noun phrases: ['Sebastian Thrun', 'self-driving cars', 'Google', 'few people', 'the company',
 'him', 'I', 'you', 'very senior CEOs', 'major American car companies', 'my hand', 'I',
 'Thrun', 'an interview', 'Recode']
Verbs: ['start', 'work', 'drive', 'take', 'tell', 'shake', 'turn', 'talk', 'say']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

[C] AIM: Named Entity recognition with diagram using NLTK corpus – treebank POS Tagging, chunking and NER.

CODE:

```
import nltk
nltk.download('treebank')
from nltk.corpus import treebank_chunk
treebank_chunk.tagged_sents()[0]
treebank_chunk.chunked_sents()[0]
treebank_chunk.chunked_sents()[0].draw()
```

OUTPUT:



PRACTICAL 6

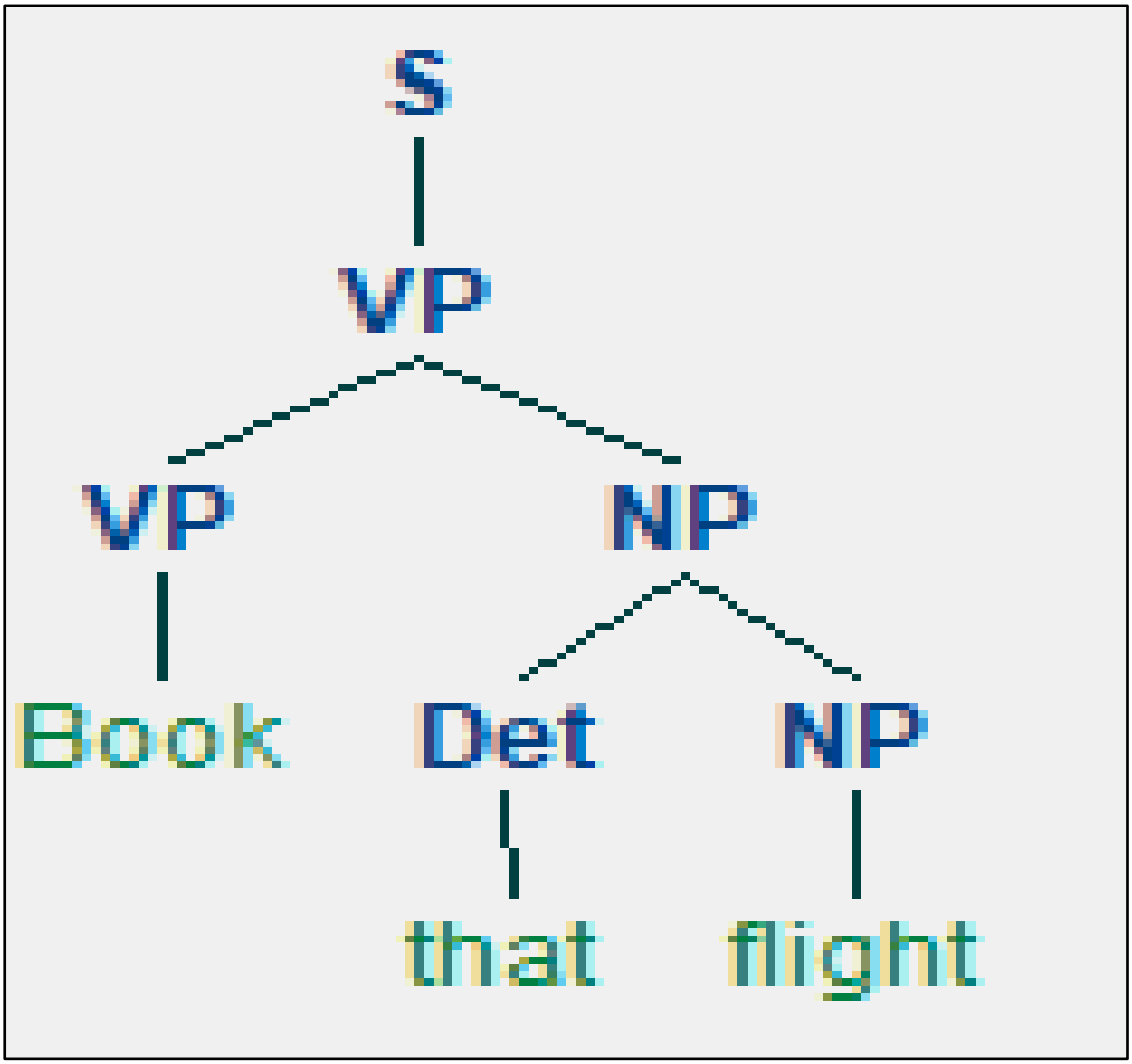
AIM: Finite state automata.

[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:



[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:

```
Accepted

Accepted

Accepted

Rejected

Rejected

Rejected

Rejected
```

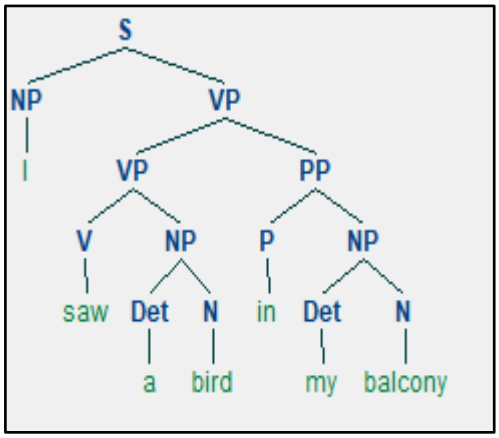
[D] Implementation of Deductive Chart Parsing using context free grammar and a given sentence.

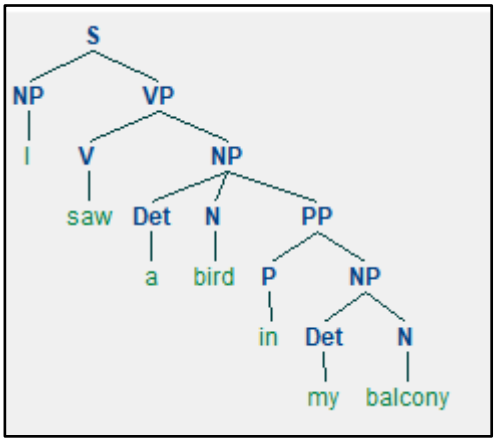
CODE:

```
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:

```
['I', 'saw', 'a', 'bird', 'in', 'my', 'balcony']
(S
  (NP I)
  (VP
    (VP (V saw) (NP (Det a) (N bird)))
    (PP (P in) (NP (Det my) (N balcony)))))
(S
  (NP I)
  (VP
    (V saw)
    (NP (Det a) (N bird) (PP (P in) (NP (Det my) (N balcony))))))
```





PRACTICAL 7

AIM: Study PorterStemmer, LancasterStemmer, RegexpStemmer, SnowballStemmer and WordNetLemmatizer.

[A] PorterStemmer.

CODE:

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

OUTPUT:

```
write
```

[B] LancasterStemmer.

CODE:

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

OUTPUT:

```
writ
```

[C] RegexpStemmer.

CODE:

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

OUTPUT:

```
writ
```

[D] SnowballStemmer.

CODE:

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

OUTPUT:

```
write
```

[E] WordNetLemmatizer.

CODE:

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

OUTPUT:

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

PRACTICAL 8

AIM: Implement Naive Bayes classifier.

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]',repl = ' ',string = 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))
```

OUTPUT:

```
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [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' 'ham']
['ham' 'ham' 'ham' ... 'ham' 'ham' 'ham']
              precision    recall  f1-score   support

         ham       0.99      0.99      0.99      1448
        spam       0.92      0.93      0.92       224

    accuracy                           0.98      1672
   macro avg       0.95      0.96      0.96      1672
weighted avg       0.98      0.98      0.98      1672

accuracy_score:  0.979066985645933
```

PRACTICAL 9

[A] AIM: Speech Tagging:

[I] Speech tagging using spacy.

CODE:

```
import spacy
spacy.cli.download("en_core_web_sm")
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)
print(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:

```
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       PROPN      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 ...
```

[II] Speech tagging using nktl.

CODE:

```
import nltk
nltk.download('state_union')
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:

```
[('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'),
 (',', ','), ('2006', 'CD'), ('THE', 'NNP'), ('PRESIDENT', 'NNP'), (':', ':'), ('Thank',
 'NNP'), ('you', 'PRP'), ('all', 'DT'), ('.', '.')]
[('Mr.', 'NNP'), ('Speaker', 'NNP'), (',', ','), ('Vice', 'NNP'), ('President', 'NNP'),
 ('Cheney', 'NNP'), (',', ','), ('members', 'NNS'), ('of', 'IN'), ('Congress', 'NNP'), (',',
 ','), ('members', 'NNS'), ('of', 'IN'), ('the', 'DT'), ('Supreme', 'NNP'), ('Court', 'NNP'),
 ('and', 'CC'), ('diplomatic', 'JJ'), ('corps', 'NN'), (',', ','), ('distinguished', 'JJ'),
 ('guests', 'NNS'), (',', ','), ('and', 'CC'), ('fellow', 'JJ'), ('citizens', 'NNS'), (':',
 ':'), ('Today', 'VB'), ('our', 'PRP$'), ('nation', 'NN'), ('lost', 'VBD'), ('a', 'DT'),
 ('beloved', 'VBN'), (',', ','), ('graceful', 'JJ'), (',', ','), ('courageous', 'JJ'),
 ('woman', 'NN'), ('who', 'WP'), ('called', 'VBD'), ('America', 'NNP'), ('to', 'TO'), ('its',
 'PRP$'), ('founding', 'NN'), ('ideals', 'NNS'), ('and', 'CC'), ('carried', 'VBD'), ('on',
 'IN'), ('a', 'DT'), ('noble', 'JJ'), ('dream', 'NN'), ('.', '.')]
```

[B] AIM: Statistical Parsing:

[I] Usage of Give and Gave in the Penn Treebank sample.

CODE:

```
#probabilitistic 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:

```
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.

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:

```
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)
```

PRACTICAL 10

[A] AIM: Multiword Expressions in NLP.

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:

```
['Good', 'cake', 'cost', 'Rs.1500\\kg', 'in', 'Mumbai', '.']
['Please', 'buy', 'me', 'one', 'of', 'them', '.']
['Thanks', '.']
```

[B] AIM: Normalized Web Distance and Word Similarity.

CODE:

```
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', 'Reliance 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
    metric="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.argmin()]
    return [centers[i] for i in clustering.labels_]
```

```
print(group_texts(texts))
```

OUTPUT:

```
['reliance', 'reliance', 'reliance', 'reliance', 'reliance', 'reliance', 'mumbai', 'mumbai',
 'mumbai', 'mumbai', 'km trading', 'km trading', 'km trading', 'km trading', 'km trading']
```

[C] AIM: Word Sense Disambiguation.

CODE:

```
#Word Sense Disambiguation
import nltk
nltk.download('wordnet')
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:

```
<bound method Synset.name of Synset('bank.n.01')>: <bound method Synset.definition of
 Synset('bank.n.01')>
<bound method Synset.name of Synset('set.n.01')>: <bound method Synset.definition of Synset
 ('set.n.01')>
<bound method Synset.name of Synset('put.v.01')>: <bound method Synset.definition of Synset
 ('put.v.01')>
```