Course Title	**Natural Language Processing lab**				**Course Type**		**HC**	
Course Code	**B21ET0604**	**Credits**	**1**		**Class**		**VI Semester**	
Course Structure	TLP	Credits	Contact Hours	Work Load	Total Number of Classes Per Semester		Assessment in Weightage	
	Theory	-	-	-				
	Practice	1	2	2	Theory	Practical	CIE	SEE
	-	-	-	-				
	Total	**1**	**2**	**2**	**-**	**26**	**25**	**25**

COURSE OVERVIEW:

The Lab offers a number of practical projects in Natural Language Processing, focusing on Text Classification, Parts of Speech processing. Some projects require previous knowledge of computational linguistics but some assume no previous background. All projects involve programming: the end result is a relatively large-scale, well-documented and efficient software package. Some of the projects may involve also some research. Python is a open-source language with a simple syntax, and a powerful set of libraries. It is widely used in many scientific areas for, NLP, string concepts.

COURSE OBJECTIVE (S):

The objectives of this course are to:

1. To learn the fundamental programs for Natural Language processing.
2. To understand the Linux OS for stopwords frequent bigrams.
3. To apply the NLP techniques for strings using python.
4. To acquire python program to remove stopwords, whitespace.
5. To implement python program for stemming a parts of speech, Lemmataization.
6. To use python program for text classification.

COURSE OUTCOMES (COs)

After the completion of the course, the student will be able to:

CO#	Course Outcomes	Pos	PSOs
CO1	Understand and execute simple Natural Language Programs.	1 to 5	1,2
CO2	Implement python program for stopwords, frequent bigrams.	1 to 5	1,2
CO3	Apply string concepts in Natural Language Processing.	1 to 5,9,12	1,2,3
CO4	Make use of python program for stemming, POS, Lemmataization and text classification.	1 to 5,9,12	1,2
CO5	Analyze and implement string concepts for various real time application using NLP.	1 to 5,9,12	1,2,3
CO6	Analyze the different statistical approaches for different types of NLP applications.	1 to 5,9,12	1,2,3

BLOOM’S LEVELOF THECOURSE OUTCOMES

CO#	Bloom’s Level					
	Remember (L1)	Understand (L2)	Apply (L3)	Analyze (L4)	Evaluate (L5)	Create (L6)
CO1			√			
CO2			√			
CO3			√			
CO4			√			
CO5				√		
CO6				√		

COURSE ARTICULATION MATRIX

CO#/ POs	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	3	1	1	1	2								1	1	
CO2	3	1	2	1	2								2	2	
CO3	3	1	2	1	2				2			2	2	2	1
CO4	3	1	2	1	2				2			2	2	2	
CO5	3	2	3	1	3				3			3	2	2	3
CO6	3	3	3	1	3				3			3	1	3	2

Note:1-Low,2-Medium,3-High

PRACTICE:

No	**Title of the Experiment**	**Tools and Techniques**	**Expected Skill /Ability**
1	Write a program to find all words that occur at least three times in the Brown Corpus.	Windows/Linux OS, IDE, Jupyter	Create and perform word count operation.
2	Write a function that finds the 50 most frequently occurring words of a text that are not stopwords.	Windows/Linux OS, IDE, Jupyter	Create and perform word count excluding stopword operation.
3	Write a program to print the 50 most frequent bigrams (pairs of adjacent words) of a text, omitting bigrams that contain stopwords.	Windows/Linux OS, IDE, Jupyter	Create and perform frequent bigram operation.
4	Write a function word_freq() that takes a word and the name of a section of the Brown Corpus as arguments, and computes the frequency of the word in that section of the corpus.	Windows/Linux OS, IDE, Jupyter	Create and perform word count operation.

5	(a) Write a program using NLTK to convert a sentence in English to French and vice-versa. (b) Define a string s = 'colorless'. Write a Python statement that changes this to "colourless" using only the slice and concatenation operations.	Windows/Linux OS, IDE, Jupyter	Create and perform slicing operation.
6	Read in some text from a corpus, tokenize it, and print the list of all wh-word types that occur. (wh-words in English are used in questions, relative clauses, and exclamations: who, which, what, and so on.) Print them in order.	Windows/Linux OS, IDE, Jupyter	Create and perform Tokenization operation.
7	Write code that removes whitespace at the beginning and end of a string, and normalizes whitespace between words to be a single-space character. a. Do this task using split() and join(). b. Do this task using regular expression substitutions.	Windows/Linux OS, IDE, Jupyter	Create and perform normalization operation.
8	Write a python program to remove stop words for a given passage from a text file using NLTK.	Windows/Linux OS, IDE, Jupyter	Implement stop word removal operation in an NLP application
9	Write a python program to implement stemming for a given sentence using NLTK.	Windows/Linux OS, IDE, Jupyter	Implement stemming operation in an NLP application
10	Write a python program to POS (Parts of Speech) tagging for the give sentence using NLTK.	Windows/Linux OS, IDE, Jupyter	Implement POS tagging operation in an NLP application
11	Write a python program to implement Lemmatization using NLTK.	Windows/Linux OS, IDE, Jupyter	Implement Lemmatization operation in an NLP application
12	Write a python program to for Text Classification for the give sentence using NLTK.	Windows/Linux OS, IDE, Jupyter	Implement Text Classification operation in an NLP application

INDEX

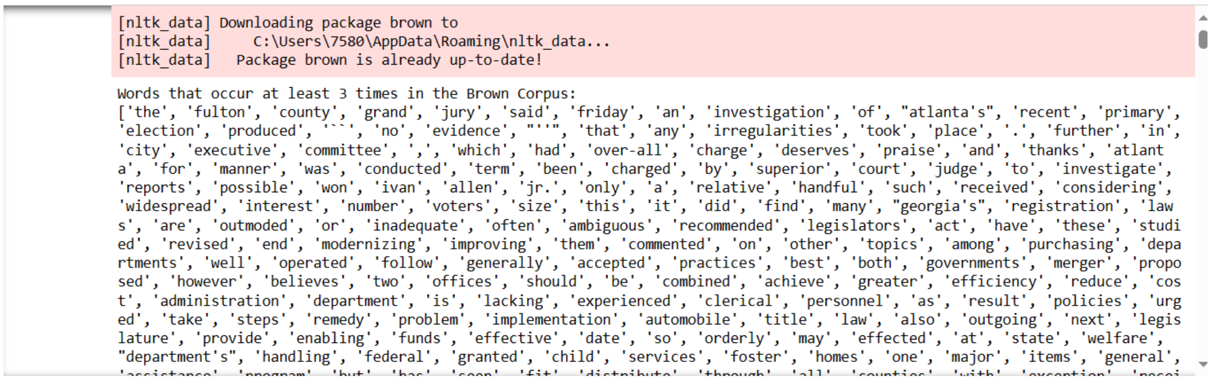
Program-1

Write a program to find all words that occur at least three times in the Brown Corpus.

```
import nltk
from nltk.corpus import brown
from collections import Counter

def find_common_words(corpus, min_occurrences):
        words = [word.lower() for word in corpus.words()]
        word_counts = Counter(words)
        common_words = [word for word, count in word_counts.items() if count >=
        min_occurrences]
        return common_words

def main():
        nltk.download('brown')
        min_occurrences = 3
        common_words = find_common_words(brown, min_occurrences)
        print(f"Words that occur at least {min_occurrences} times in the Brown
        Corpus:")
        print(common_words)

if __name__ == "__main__":
    main()
```

OUTPUT

Jupyter Notebook



```
[nltk_data] Downloading package brown to
[nltk_data]     C:\Users\7580\AppData\Roaming\nltk_data...
[nltk_data]   Package brown is already up-to-date!

Words that occur at least 3 times in the Brown Corpus:
['the', 'fulton', 'county', 'grand', 'jury', 'said', 'friday', 'an', 'investigation', 'of', "atlanta's", 'recent', 'primary',
'election', 'produced', '``', 'no', 'evidence', "''", 'that', 'any', 'irregularities', 'took', 'place', '.', 'further', 'in',
'city', 'executive', 'committee', ',', 'which', 'had', 'over-all', 'charge', 'deserves', 'praise', 'and', 'thanks', 'atlant
a', 'for', 'manner', 'was', 'conducted', 'term', 'been', 'charged', 'by', 'superior', 'court', 'judge', 'to', 'investigate',
'reports', 'possible', 'won', 'ivan', 'allen', 'jr.', 'only', 'a', 'relative', 'handful', 'such', 'received', 'considering',
'widespread', 'interest', 'number', 'voters', 'size', 'this', 'it', 'did', 'find', 'many', "georgia's", 'registration', 'law
s', 'are', 'outmoded', 'or', 'inadequate', 'often', 'ambiguous', 'recommended', 'legislators', 'act', 'have', 'these', 'studi
ed', 'revised', 'end', 'modernizing', 'improving', 'them', 'commented', 'on', 'other', 'topics', 'among', 'purchasing', 'depa
rtments', 'well', 'operated', 'follow', 'generally', 'accepted', 'practices', 'best', 'both', 'governments', 'merger', 'propo
sed', 'however', 'believes', 'two', 'offices', 'should', 'be', 'combined', 'achieve', 'greater', 'efficiency', 'reduce', 'cos
t', 'administration', 'department', 'is', 'lacking', 'experienced', 'clerical', 'personnel', 'as', 'result', 'policies', 'urg
ed', 'take', 'steps', 'remedy', 'problem', 'implementation', 'automobile', 'title', 'law', 'also', 'outgoing', 'next', 'legis
lature', 'provide', 'enabling', 'funds', 'effective', 'date', 'so', 'orderly', 'may', 'effected', 'at', 'state', 'welfare',
"department's", 'handling', 'federal', 'granted', 'child', 'services', 'foster', 'homes', 'one', 'major', 'items', 'general',
'assistance', 'program', 'but', 'has', 'seen', 'fit', 'distribute', 'through', 'all', 'counties', 'with', 'exception', 'recei
```

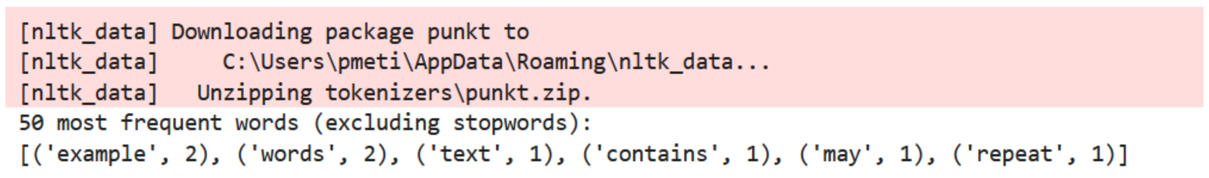
Program-2

Write a function that finds the 50 most frequently occurring words of a text that are not stopwords.

```
import nltk
from nltk.corpus import stopwords
from nltk import FreqDist
from nltk.tokenize import word_tokenize
nltk.download('punkt')
nltk.download('stopwords')
def most_frequent_words(text):
      # Tokenize the text into words
      words = word_tokenize(text)
      # Filter out stopwords
      stop_words = set(stopwords.words('english'))
      filtered_words = [word.lower() for word in words if word.isalpha() and
      word.lower() not in stop_words]
# Calculate the frequency distribution of words
freq_dist = FreqDist(filtered_words)
# Get the 50 most frequent words
most_frequent = freq_dist.most_common(50)
return most_frequent
# Example usage:
sample_text = "This is an example text. It contains some words, and some of these
words may repeat. This is just an example."
result = most_frequent_words(sample_text)
# Display the result
print("50 most frequent words (excluding stopwords):")
print(result)
```

OUTPUT

Jupyter Notebook



```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\pmeti\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping tokenizers\punkt.zip.
50 most frequent words (excluding stopwords):
[('example', 2), ('words', 2), ('text', 1), ('contains', 1), ('may', 1), ('repeat', 1)]
```

Program-3

Write a program to print the 50 most frequent bigrams (pairs of adjacent words) of a text, omitting bigrams that contain stopwords. Windows/Linux OS, IDE, Jupyter. Create and perform frequent bigram operations.

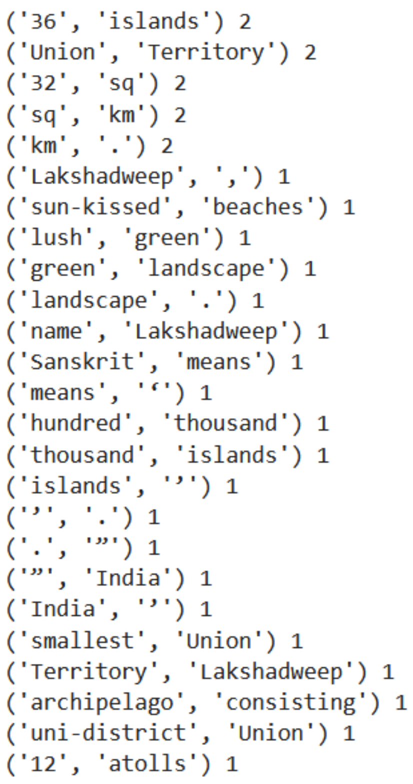
```
import nltk
from nltk.corpus import stopwords
from nltk import FreqDist
from nltk import bigrams

# Download NLTK resources (stopwords)
nltk.download('stopwords')

def get_frequent_bigrams(text, N=50):
	# Tokenize the text into words
	words = nltk.word_tokenize(text)

	# Create a list of English stopwords
	stop_words = set(stopwords.words('english'))

	# Filter out stopwords and create bigrams
	filtered_bigrams = [bigram for bigram in bigrams(words) if all(word.lower()
	not in stop_words for word in bigram)]

	# Calculate the frequency distribution of bigrams
	freq_dist = FreqDist(filtered_bigrams)

	# Get the N most common bigrams
	most_common_bigrams = freq_dist.most_common(N)
	return most_common_bigrams

# Example text
example_text = """
Lakshadweep, the group of 36 islands is known for its exotic and sun-kissed beaches and lush
green landscape. The name Lakshadweep in Malayalam and Sanskrit means ‘a hundred
thousand islands’. India’s smallest Union Territory Lakshadweep is an archipelago consisting
```

of 36 islands with an area of 32 sq km. It is a uni-district Union Territory and comprises of 12 atolls, three reefs, five submerged banks and ten inhabited islands. The islands have a total area of 32 sq km. The capital is Kavaratti and it is also the principal town of the UT. All Islands are 220 to 440 km away from the coastal city of Kochi in Kerala, in the emerald Arabian Sea. he natural landscapes, the sandy beaches, abundance of flora and fauna and the absence of a rushed lifestyle enhance the mystique of Lakshadweep.

```
"""


# Call the function and print the 50 most frequent bigrams omitting stopwords
result = get_frequent_bigrams(example_text, N=50)
for bigram, frequency in result:
        print(bigram, frequency)
```

<u>OUTPUT</u>

Jupyter Notebook



```
('36', 'islands') 2
('Union', 'Territory') 2
('32', 'sq') 2
('sq', 'km') 2
('km', '.') 2
('Lakshadweep', ',') 1
('sun-kissed', 'beaches') 1
('lush', 'green') 1
('green', 'landscape') 1
('landscape', '.') 1
('name', 'Lakshadweep') 1
('Sanskrit', 'means') 1
('means', '‘') 1
('hundred', 'thousand') 1
('thousand', 'islands') 1
('islands', '’') 1
('’', '.') 1
('.', '”') 1
('”', 'India') 1
('India', '’') 1
('smallest', 'Union') 1
('Territory', 'Lakshadweep') 1
('archipelago', 'consisting') 1
('uni-district', 'Union') 1
('12', 'atolls') 1
```

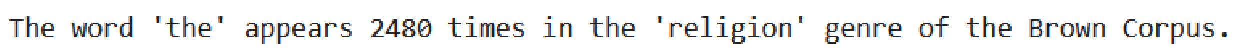
Program-4

Write a function word_freq() that takes a word and the name of a section of the Brown Corpus as arguments, and computes the frequency of the word in that section of the corpus.

```
import nltk
from nltk.corpus import brown

def word_freq(word, genre):
      fdist = nltk.FreqDist([w.lower() for w in
      nltk.corpus.brown.words(categories=genre)])
      print(f"The word '{word}' appears {fdist[word]} times in the '{genre}'
      genre of the Brown Corpus.")

# Example usage:
word_freq('the', 'religion')
```

OUTPUT

Jupyter Notebook



```
The word 'the' appears 2480 times in the 'religion' genre of the Brown Corpus.
```

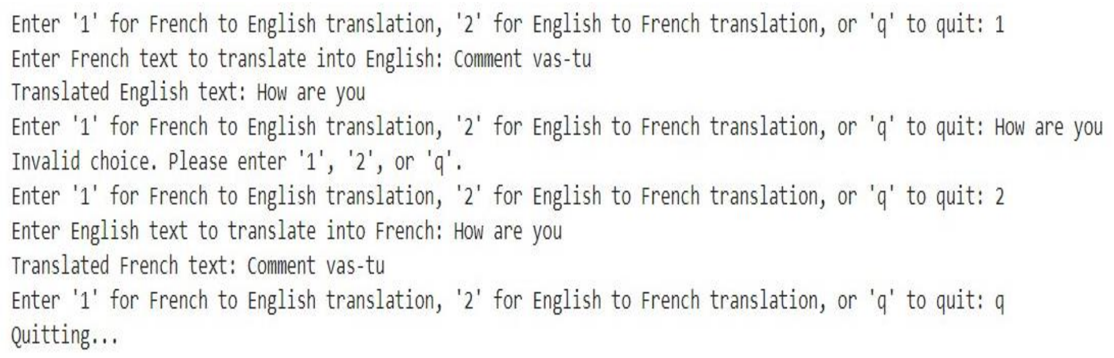
Program-5

(a) Write a prog ram using NLTK to convert a sentence in English to French and vice-versa.

```
pip install googletrans==4.0.0-rc1

import nltk

from googletrans import Translator

def translate_text(text, src_lang, dest_lang):

    translator = Translator()

    translation = translator.translate(text, src=src_lang, dest=dest_lang)

    translated_text = translation.text

    return translated_text

def french_to_english(french_text):

    french_tokens = nltk.word_tokenize(french_text)

    french_text_cleaned = ' '.join(french_tokens)

    english_translation = translate_text(french_text_cleaned, 'fr', 'en')

    return english_translation

def english_to_french(english_text):

    french_translation = translate_text(english_text, 'en', 'fr')

    return french_translation

if __name__ == "__main__":

    while True:

        choice = input("Enter '1' for French to English translation, '2' for
English to French translation, or 'q' to quit: ")


        if choice == '1':

            french_text = input("Enter French text to translate into English: ")

            english_translation = french_to_english(french_text)

            print("Translated English text:", english_translation)

        elif choice == '2':

            english_text = input("Enter English text to translate into French: ")

            french_translation = english_to_french(english_text)

            print("Translated French text:", french_translation)

        elif choice.lower() == 'q':

            print("Quitting...")
            break
        else:
            print("Invalid choice. Please enter '1', '2', or 'q'.")
```

OUTPUT

Jupyter Notebook



```
Enter '1' for French to English translation, '2' for English to French translation, or 'q' to quit: 1
Enter French text to translate into English: Comment vas-tu
Translated English text: How are you
Enter '1' for French to English translation, '2' for English to French translation, or 'q' to quit: How are you
Invalid choice. Please enter '1', '2', or 'q'.
Enter '1' for French to English translation, '2' for English to French translation, or 'q' to quit: 2
Enter English text to translate into French: How are you
Translated French text: Comment vas-tu
Enter '1' for French to English translation, '2' for English to French translation, or 'q' to quit: q
Quitting...
```

Program-5

(b) Define a string s = 'colorless'. Write a Python statement that changes this to "colourless" using only the slice and concatenation operations. Windows/Linux OS, IDE, JupyterCreate and perform slicing operation.

```
# Define the original string

s = 'colorless'

modified_s = s[:3] + 'our' + s[5:]

# Print the modified string

print(modified_s)
```

<u>OUTPUT</u>

<u>Jupyter Notebook</u>

```
colourless
```

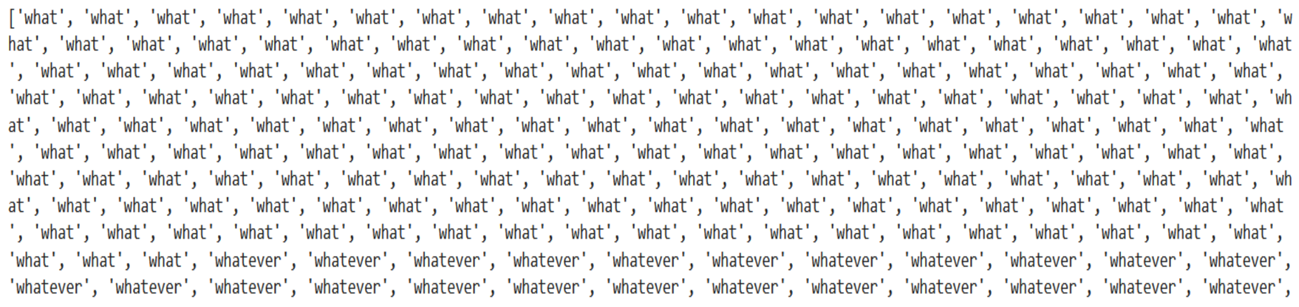
Program-6

Read in some text from a corpus, tokenize it, and print the list of all wh-word types that occur. (Wh-words in English are used in questions, relative clauses and exclamations: who, which, what, and so on.) Print them in order. Are any words duplicated in this list, because of the presence of case distinctions or punctuation.

```
import nltk
from nltk import word_tokenize

f = open('corpus.txt')
raw = f.read()

# tokenizes that text
tokens = word_tokenize(raw)

# pulls out all the words that start with wh. and prints it out.
wh_words = [word for word in tokens if word.startswith('wh')]

# sorts the list and prints it
wh_words.sort()
print(wh_words)
```

<u>OUTPUT</u>

Jupyter Notebook



```
['what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'w
hat', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what
', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what',
'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'wh
at', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what
', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what',
'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'wh
at', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what
', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what', 'what',
'what', 'what', 'what', 'whatever', 'whatever', 'whatever', 'whatever', 'whatever', 'whatever', 'whatever', 'whatever', 'whatever', 'whatever', 'whatever',
'whatever', 'whatever', 'whatever', 'whatever', 'whatever', 'whatever', 'whatever', 'whatever', 'whatever', 'whatever', 'whatever', 'whatever', 'whatever',
```

Program-7

Write code that removes whitespace at the beginning and end of a string and normalizes whitespace between words to be a single space character.

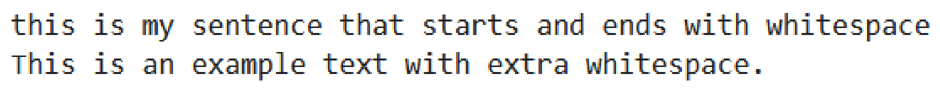
# 1. do this task using split () and join ()

# 2. do this task using regular expression substitutions.

```
import re

sent = ' this is my sentence that starts and ends with whitespace '

pattern = re.compile('^\s|\s$')

sent = pattern.sub('', sent)

print(sent)

sent = "   This    is    an    example   text   with   extra   whitespace.    "

sent = sent.split(' ')

sent = [word for word in sent if word]

sent = ' '.join(sent)

print(sent)
```

OUTPUT

Jupyter Notebook



```
this is my sentence that starts and ends with whitespace
This is an example text with extra whitespace.
```

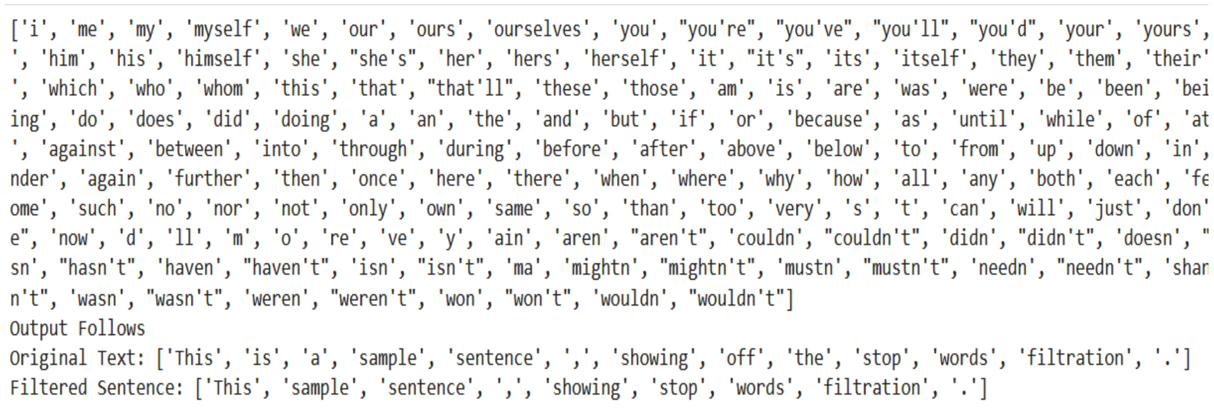
Program-8

Write a python program to remove stop words for a given passage from a text file using NLTK. Implement stop word removal operation in an NLP application.

```
from nltk.corpus import stopwords

from nltk.tokenize import word_tokenize

example_sent = """This is a sample sentence,

                          showing off the stop words filtration."""

stop_words = set(stopwords.words('english'))

print(stopwords.words('english'))

word_tokens = word_tokenize(example_sent)

filtered_sentence = [w for w in word_tokens if not w.lower() in stop_words]

#with no lower case conversion

filtered_sentence = []

for w in word_tokens:

      if w not in stop_words:

            filtered_sentence.append(w)

print("Output Follows")

print("Original Text:",word_tokens)

print("Filtered Sentence:",filtered_sentence)
```

OUTPUT

Jupyter Notebook



```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours',
', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their'
', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'bei
ing', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at
', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in',
nder', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'fe
ome', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don'
e", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "
sn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan
n't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
Output Follows
Original Text: ['This', 'is', 'a', 'sample', 'sentence', ',', 'showing', 'off', 'the', 'stop', 'words', 'filtration', '.']
Filtered Sentence: ['This', 'sample', 'sentence', ',', 'showing', 'stop', 'words', 'filtration', '.']
```

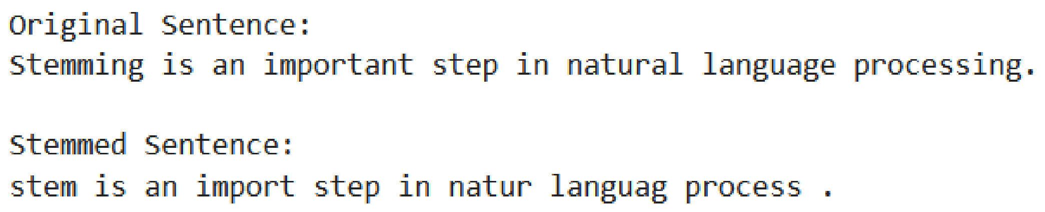
Program-9

Write a python program to implement stemming for a given sentence using NLTK? Windows/Linux OS, IDE, Jupyter Implement stemming operation in an NLP application.

```
import nltk

from nltk.stem import PorterStemmer

from nltk.tokenize import word_tokenize


def apply_stemming(sentence):

    # Tokenize the sentence into words

    words = word_tokenize(sentence)


    # Initialize the Porter Stemmer

    porter_stemmer = PorterStemmer()


    # Apply stemming to each word

    stemmed_words = [porter_stemmer.stem(word) for word in words]


    # Join the stemmed words back into a sentence

    stemmed_sentence = ' '.join(stemmed_words)


    return stemmed_sentence


def main():

    # Example sentence

    input_sentence = "Stemming is an important step in natural language
processing."


    # Apply stemming to the sentence

    stemmed_result = apply_stemming(input_sentence)


    # Display the results

    print("Original Sentence:")
    print(input_sentence)
    print("\nStemmed Sentence:")
    print(stemmed_result)

if __name__ == "__main__":
    main()
```

OUTPUT

Jupyter Notebook



```
Original Sentence:
Stemming is an important step in natural language processing.

Stemmed Sentence:
stem is an import step in natur languag process .
```

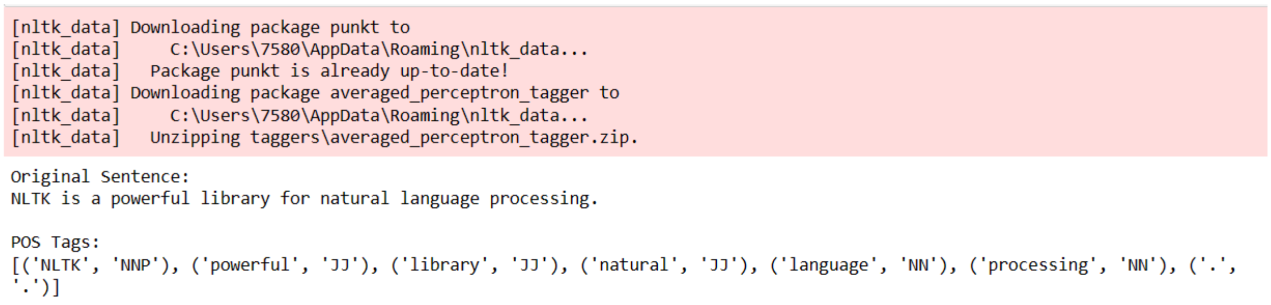
Program-10

Write a python program to POS (Parts of Speech) tagging for the give sentence using NLTK. Implement POS tagging operation in an NLP application.

```
import nltk

from nltk import pos_tag

from nltk.tokenize import word_tokenize

from nltk.corpus import stopwords


nltk.download('punkt')

nltk.download('averaged_perceptron_tagger')


def pos_tagging(sentence):

    # Tokenize the sentence into words

    words = word_tokenize(sentence)


    # Remove stopwords (optional, depending on your use case)

    stop_words = set(stopwords.words('english'))

    words = [word for word in words if word.lower() not in stop_words]


    # Perform POS tagging

    pos_tags = pos_tag(words)


    return pos_tags


def main():

    # Example sentence

    input_sentence = "NLTK is a powerful library for natural language processing."


    # Perform POS tagging on the sentence

    pos_tags_result = pos_tagging(input_sentence)


    # Display the results

    print("Original Sentence:")
    print(input_sentence)

    print("\nPOS Tags:")

    print(pos_tags_result)


if __name__ == "__main__":

    main()
```

OUTPUT

Jupyter Notebook



```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\7580\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\7580\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping taggers\averaged_perceptron_tagger.zip.

Original Sentence:
NLTK is a powerful library for natural language processing.

POS Tags:
[('NLTK', 'NNP'), ('powerful', 'JJ'), ('library', 'JJ'), ('natural', 'JJ'), ('language', 'NN'), ('processing', 'NN'), ('.',
'.')]
```

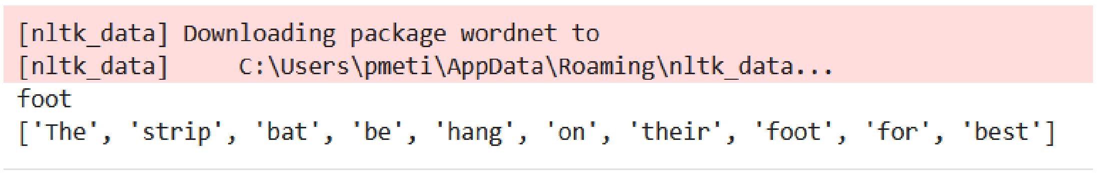
Program-11

Write a python program to POS (Parts of Speech) tagging for the give sentence using NLTK. Implement POS tagging operation in an NLP application

```
import nltk

nltk.download('wordnet')

from nltk.corpus import wordnet

from nltk.stem import WordNetLemmatizer


def get_wordnet_pos(word):

    """Map POS tag to first character lemmatize() accepts"""

    tag = nltk.pos_tag([word])[0][1][0].upper()

    tag_dict = {"J": wordnet.ADJ,

                "N": wordnet.NOUN,

                "V": wordnet.VERB,

                "R": wordnet.ADV}


    return tag_dict.get(tag, wordnet.NOUN)


# 1. Initialize the Lemmatizer

lemmatizer = WordNetLemmatizer()


# 2. Lemmatize Single Word with the appropriate POS tag

word = 'feet'

print(lemmatizer.lemmatize(word, get_wordnet_pos(word)))


# 3. Lemmatize a Sentence with the appropriate POS tag

sentence = "The striped bats are hanging on their feet for best"

print([lemmatizer.lemmatize(w, get_wordnet_pos(w)) for w in
nltk.word_tokenize(sentence)])
```

OUTPUT

Jupyter Notebook



```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\pmeti\AppData\Roaming\nltk_data...
foot
['The', 'strip', 'bat', 'be', 'hang', 'on', 'their', 'foot', 'for', 'best']
```

Program-12

Write a python program to for Text Classification for the give sentence using NLTK. Implement Text Classification operation in an NLP application

```
import nltk

from nltk.corpus import movie_reviews

from nltk import FreqDist

from nltk import classify

from nltk import NaiveBayesClassifier


# Download the movie_reviews dataset if not already downloaded

nltk.download('movie_reviews')


# Feature extraction function

def extract_features(words):

    return dict(FreqDist(words))


# Prepare the dataset

documents = [(list(movie_reviews.words(fileid)), category)

             for category in movie_reviews.categories()

             for fileid in movie_reviews.fileids(category)]


# Shuffle the documents

import random

random.shuffle(documents)


# Split the dataset into training and testing sets

split_ratio = int(len(documents) * 0.8)

train_set, test_set = documents[:split_ratio], documents[split_ratio:]


# Extract features using the defined function

training_features = [(extract_features(words), category) for (words, category) in
train_set]

testing_features = [(extract_features(words), category) for (words, category) in
test_set]
```

```
# Train the Naive Bayes classifier
classifier = NaiveBayesClassifier.train(training_features)

# Evaluate the classifier on the testing set
accuracy = classify.accuracy(classifier, testing_features)
print("Accuracy:", accuracy)

# Example sentences to classify
new_sentences = [
    "This movie was fantastic!",
    "I didn't like the plot of this film.",
    "The acting was superb in this movie.",
    "The screenplay was terrible."
]

# Classify the new sentences
for sentence in new_sentences:
    words = nltk.word_tokenize(sentence)
    features = extract_features(words)
    category = classifier.classify(features)
    print(f"Predicted category for '{sentence}': {category}")
```

OUTPUT

Jupyter Notebook

```
[nltk_data] Downloading package movie_reviews to
[nltk_data]     C:\Users\pmeti\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\movie_reviews.zip.
Accuracy: 0.7675
Predicted category for 'This movie was fantastic!': pos
Predicted category for 'I didn't like the plot of this film.': neg
Predicted category for 'The acting was superb in this movie.': pos
Predicted category for 'The screenplay was terrible.': neg
```

Rubrics

Program or Lab internals – 13

CIA – 8

- P – 3
- R – 2
- V – 2
- A - 1

Lab Internal - 5

Mini project – 12

- Execution - 6
- Publication/Project Competition - 6