Aim: Write a program to implement Sentence Segmentation and Word Tokenization.
Source Code:

```
#word Tokenization
from nltk.tokenize import word_tokenize
str=input("Enter Info/Data for tokenization:")
print(word_tokenize(str))
print("Number of tokens for word:",len(str))


#Sentence Tokenization
from nltk.tokenize import sent_tokenize
str1=sent_tokenize(str)
print(str1)
print("Number of tokens for sentence:",len(str1))
```

Output:

```
Enter Info/Data for tokenization: India is my country. All indians are my brothers and sis
['India', 'is', 'my', 'country', '.', 'All', 'indians', 'are', 'my', 'brothers', 'and', 's
Number of tokens for word: 80
['India is my country.', 'All indians are my brothers and sisters.', 'I love my Country.']
Number of tokens for sentence: 3
```

2 Write a program to Implement stemming and lemmatization

Stemming
Program 1:

```
import nltk
from nltk.stem import PorterStemmer
# Initialize Python porter stemmer
ps = PorterStemmer()
# Example inflections to reduce
example_words = ["program","programming","programer","programs","programmed"]
# Perform stemming
print("{0:20}{1:20}".format("--Word--","--Stem--"))
for word in example_words:
  print ("{0:20}{1:20}".format(word, ps.stem(word)))
```

Program 2:

```
import string
from nltk.tokenize import word_tokenize
example_sentence = "Python programmers often tend like programming in python because it's
# Remove punctuation
example_sentence_no_punct = example_sentence.translate(str.maketrans("", "", string.punctu
# Create tokens
```

```
word_tokens = word_tokenize(example_sentence_no_punct)
# Perform stemming
print("{0:20}{1:20}".format("--Word--","--Stem--"))
for word in word_tokens:
   print ("{0:20}{1:20}".format(word, ps.stem(word)))
```

Lemmatization

Program 1:

```
from nltk.stem import WordNetLemmatizer
nltk.download("wordnet")
nltk.download("omw-1.4")
# Initialize wordnet lemmatizer
wnl = WordNetLemmatizer()
# Example inflections to reduce
example_words = ["program","programming","programer","programs","programmed"]
# Perform lemmatization
print("{0:20}{1:20}".format("--Word--","--Lemma--"))
for word in example_words:
  print ("{0:20}{1:20}".format(word, wnl.lemmatize(word, pos="v")))
```

Program 2:

```
example_sentence = "Python programmers often tend like programming in python because it's
# Remove punctuation
example_sentence_no_punct = example_sentence.translate(str.maketrans("", "", string.punctu
word_tokens = word_tokenize(example_sentence_no_punct)
# Perform lemmatization
print("{0:20}{1:20}".format("--Word--","--Lemma--"))
for word in word_tokens:
  print ("{0:20}{1:20}".format(word, wnl.lemmatize(word, pos="v")))
```

3 Write a program to Implement a tri-gram model

```
Import nltk
from nltk.util import ngrams
```

```
from nltk.util import ngrams


n=3
Sentence="You will face many defeats in life,but never let it go"
trigrams = ngrams(Sentence.split(),n)


for item in trigrams:
   print(item)
```

4 Write a program to Implement PoS tagging using HMM & Neural Model

Aim: Write a program to implement POS Tagging using HMM.

Source Code:

```
import nltk
import numpy as np
import pandas as pd
import random
from sklearn .model_selection import train_test_split
import pprint,time
#download the treebank corpus from nltk
nltk.download('treebank')
#download the universal tagset from nltk
nltk.download('universal_tagset')
#reading the treebank tagged sentences
nltk_data=list(nltk.corpus.treebank.tagged_sents(tagset='universal'))
#print the first two sentences along with tags
print(nltk_data[:2])
```

Output:

Aim: Write a program to implement POS Tagging using Neural Model

```
import nltk
nltk.download('treebank')
nltk.download('brown')
nltk.download('conll2000')
#load POS tagged corpara from nltk
treebank_corpus=nltk.corpus.treebank.tagged_sents(tagset='univrsal')
brown_corpus=nltk.corpus.brown.tagged_sents(tagset='universal')
conll_corpus=nltk.corpus.conll2000.tagged_sents(tagset='universal')
#merging the dataframes to create a master df
tagged_sentences=treebank_corpus+brown_corpus+conll_corpus
print(tagged_sentences)
```

Output:

New Section

5 Write a program to Implement syntactic parsing of a given text

Aim: Write A Program To Implement Syntactic Parsing Of A Given Text.
Source Code:

```
#Import required libraries
import nltk
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
from nltk import pos_tag,word_tokenize,RegexpParser
#Example text
sample_text="The little yellow dog barked at the cat"
#Find all parts of speech in above sentence
tagged=pos_tag(word_tokenize(sample_text))
#Extract all parts of speech from any text
chunker =RegexpParser("""             NP:{<DT>?<JJ>*<NN>}
                                      P:{<IN>}
                                      V:{<V.*>}
                                      PP:{<P> <NP>}
                                      VP:{<V> <NP|PP>*}
                      """)
#print all parts of speech in above sentence.
output = chunker.parse(tagged)
print("After Extracting\n",output)
output.draw()
```

Write a program to Implement dependency parsing of a given text

Aim: 6 Write a program to implement dependency parsing of a given text.
Source Code:

```
import spacy
from spacy import displacy
#load the language model
nlp = spacy.load("en_core_web_sm")
sentence = 'Deemed universities charage hug fees'
#nlp function returns an object with individual token information
#linnguistic features and relationships
doc = nlp(sentence)
print("{:<15}|{:<8}|{:<15}|{:<20}".format('Token','Relation','Head','Children'))
print("-" * 70)
for token in doc:
    #print the token,dependency nature,head and all dependencies of the token
    print("{:<15}|{:<8}|{:<15}|{:<20}"
          .format(str(token.text),str(token.dep_),str(token.head.text),str([child for chil
#use displayCy to visualize the dependencies
    displacy.render(doc,style='dep',jupyter=True,options={'distance':120})
```

Output:

Aim: 7 Write a program to implement named Entity Recognition(NER).
Source Code:

```
import spacy
from spacy import displacy
NER = spacy.load("en_core_web_sm")
raw_text2="The Indian Space Reasearch Organisation or is the national space agency."
text1 = NER(raw_text2)
for word in text1.ents:
    print(word.text,word.label)
#use displayCy to visualize the dependencies
    displacy.render(text1,style='ent',jupyter=True)
```

Aim: 8 Write a program to Implement Text Summarization for the given sample text
Source Code:

```
from nltk.corpus import stopwords
from nltk.cluster.util import cosine_distance
import numpy as np
import networkx as nx
import nltk
nltk.download("stopwords")
def read_article(file_name):
    file = open(file_name, "r")
    filedata = file.readlines()
    article = filedata[0].split(". ")
    sentences = []
    for sentence in article:
        print(sentence)
        sentences.append(sentence.replace("[^a-zA-Z]", " ").split(" "))
    sentences.pop()
    return sentences
def sentence_similarity(sent1, sent2, stopwords=None):
    if stopwords is None:
        stopwords = []
    sent1 = [w.lower() for w in sent1]
    sent2 = [w.lower() for w in sent2]
    all_words = list(set(sent1 + sent2))
    vector1 = [0] * len(all_words)
    vector2 = [0] * len(all_words)
    # build the vector for the first sentence
    for w in sent1:
        if w in stopwords:
            continue
        vector1[all_words.index(w)] += 1
    # build the vector for the second sentence
    for w in sent2:
```

```
            if w in stopwords:
                continue
            vector2[all_words.index(w)] += 1
        return 1 - cosine_distance(vector1, vector2)
    def build_similarity_matrix(sentences, stop_words):
        # Create an empty similarity matrix
        similarity_matrix = np.zeros((len(sentences), len(sentences)))
        for idx1 in range(len(sentences)):
            for idx2 in range(len(sentences)):
                if idx1 == idx2: #ignore if both are same sentences
                    continue
                similarity_matrix[idx1][idx2] = sentence_similarity(sentences[idx1], sentences
        return similarity_matrix
    def generate_summary(file_name, top_n=5):
        stop_words = stopwords.words('english')
        summarize_text = []
        # Step 1 - Read text anc split it
        sentences =  read_article(file_name)
        # Step 2 - Generate Similary Martix across sentences
        sentence_similarity_martix = build_similarity_matrix(sentences, stop_words)
        # Step 3 - Rank sentences in similarity martix
        sentence_similarity_graph = nx.from_numpy_array(sentence_similarity_martix)
        scores = nx.pagerank(sentence_similarity_graph)
        # Step 4 - Sort the rank and pick top sentences
        ranked_sentence = sorted(((scores[i],s) for i,s in enumerate(sentences)), reverse=True
        print("Indexes of top ranked_sentence order are ", ranked_sentence)
        for i in range(top_n):
          summarize_text.append(" ".join(ranked_sentence[i][1]))
        # Step 5 - Offcourse, output the summarize texr
        print("Summarize Text: \n", ". ".join(summarize_text))
    # let's begin
    generate_summary( "input1.txt", 2)
```

output:

Colab paid products - Cancel contracts here