Name: Avinash R. Kauran, Roll No: CS22006
Course: Natural Language Processing

PRACTICAL NO : 01

AIM: Write a program to implement Sentence Segmentation & WordTokenization

THEORY:
Tokenization is used in natural language processing to split paragraphs and sentences into smaller units that can be more easily assigned meaning.

Sentence Tokenization
Sentence tokenization is the process of splitting text into individual sentences.

Word Tokenization
Word tokenization is the most common version of tokenization. It takes natural breaks,like pauses in speech or spaces in text, and splits the data into its respective words using delimiters (characters like ‘,’ or ‘;’ or ‘“,”’). While this is the simplest way to separate speech or text into its parts.

Modules
NLTK contains a module called tokenize() which further classifies into two sub-categories:

- ✓ Word tokenize: We use the word_tokenize() method to split a sentence intotokens or words.
- ✓ Sentence tokenize: We use the sent_tokenize() method to split a document orparagraph into sentences

Source Code:

```
import nltk
from nltk.tokenize import word_tokenize
```

```
#words in file are "This is start of new Era"
with open('/content/sample_data/hello.txt') as sente:
    lines = sente.readlines()
```

```
for content in lines:
    line = nltk.sent_tokenize(content)
    print(content)
    print("Tokens:", word_tokenize(content))
```

Output:

```
This is start of new Era
Tokens: ['This', 'is', 'start', 'of', 'new', 'Era']
```

Name: Avinash R. Kauran, Roll No: CS22006
Course: Natural Language Processing

PRACTICAL NO : 02

AIM: Write a program to Implement Stemming & Lemmatization.

THEORY:
What is Stemming?
Stemming is a technique used to extract the base form of the words by removing affixesfrom them. It is just like cutting down the branches of a tree to its stems. For example, the stem of the words eating, eats, eaten is eat.

Search engines use stemming for indexing the words. That’s why rather than storing allforms of a word, a search engine can store only the stems. In this way, stemming reduces the size of the index and increases retrieval accuracy.

Modules
NLTK has PorterStemmer class with the help of which we can easily implement PorterStemmer algorithms for the word we want to stem. This class knows several regular word forms and suffixes with the help of which it can transform the input word to a final stem.

NLTK has LancasterStemmer class with the help of which we can easily implementLancaster Stemmer algorithms for the word we want to stem.

NLTK has SnowballStemmer class with the help of which we can easily implement Snowball Stemmer algorithms. It supports 15 non-English languages. In order to use this steaming class, we need to create an instance with the name of the language we areusing and then call the stem() method.

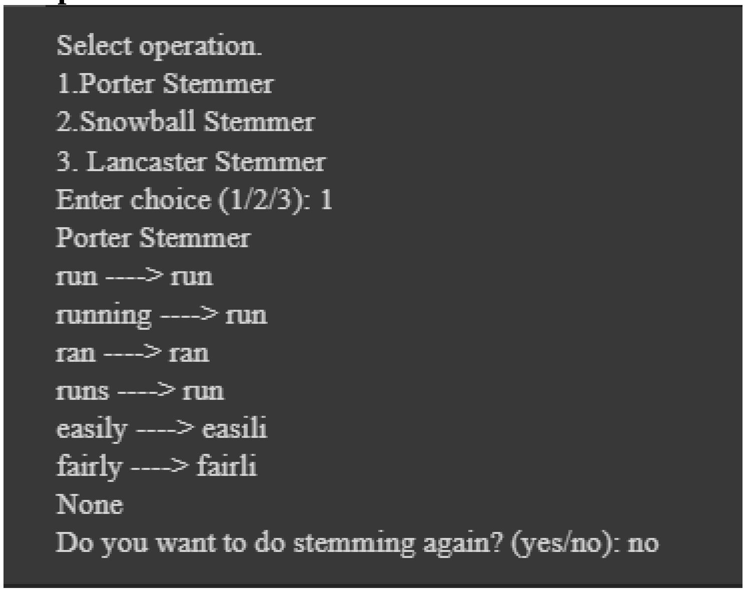
What is Lemmatization?
Lemmatization is the process of grouping together the different inflected forms of aword so they can be analyzed as a single item.
Lemmatization is similar to stemming but it brings context to the words. So it linkswords with similar meanings to one word.
Text preprocessing includes both Stemming as well as Lemmatization.
Many times, people find these two terms confusing. Some treat these two as the same.Actually, lemmatization is preferred over Stemming because lemmatization does morphological analysis of the words.

Applications of lemmatization are:
- ✓ Used in comprehensive retrieval systems like search engines.
- ✓ Used in compact indexing

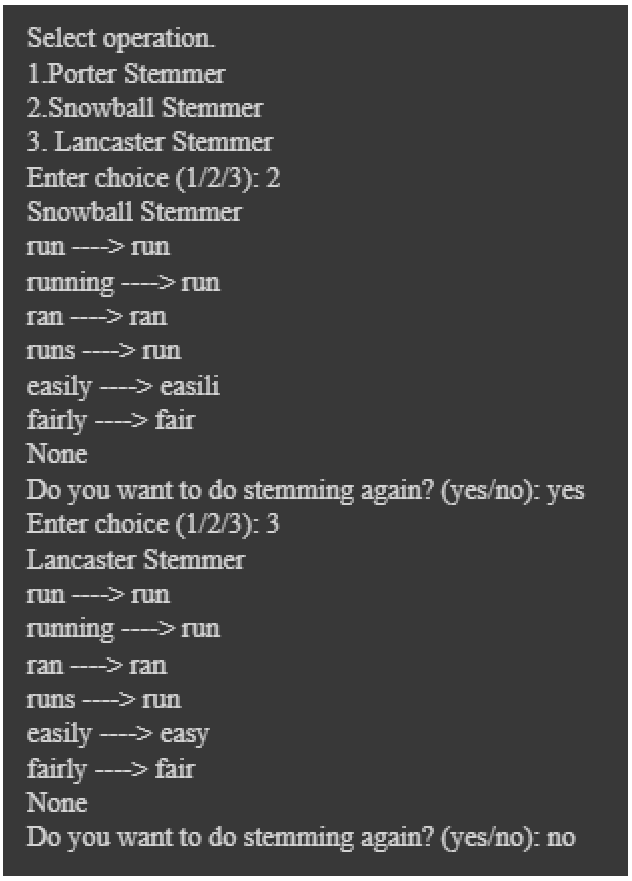
STEMMING:

Source Code:
```
import nltk
nltk.download('averaged_perceptron_tagger')
from nltk.stem import PorterStemmer
from nltk.stem import SnowballStemmer
from nltk.stem import LancasterStemmer

words = ['run', 'running', 'ran', 'runs', 'easily', 'fairly']

def portstem(words):
    ps = PorterStemmer()
    print("Porter Stemmer")
    for word in words:
        print(word, "---->", ps.stem(word))
# print(portstem(words))

def snowstem(words):
    ss = SnowballStemmer(language='english')
    print("Snowball Stemmer")
    for word in words:
        print(word, "---->", ss.stem(word))
# print(snowstem(words))

def lanstem(words):
    ss = LancasterStemmer()
    print("Lancaster Stemmer")
    for word in words:
        print(word, "---->", ss.stem(word))
# print(lanstem)

print("Select operation.")
print("1.Porter Stemmer")
print("2.Snowball Stemmer")
print("3. Lancaster Stemmer")
while True:
  choice = input("Enter choice (1/2/3): ")
  if choice in ('1', '2', '3'):
    if choice == '1':
      print(portstem(words))
    elif choice == '2':
      print(snowstem(words))
    elif choice == '3':
      print(lanstem (words))
    next_calculation = input("Do you want to do stemming again? (yes/no): ")
    if next_calculation == "no":
      break
  else:
    print("Invalid Input")
```

Output:



```
Select operation.
1.Porter Stemmer
2.Snowball Stemmer
3. Lancaster Stemmer
Enter choice (1/2/3): 1
Porter Stemmer
run ----> run
running ----> run
ran ----> ran
runs ----> run
easily ----> easili
fairly ----> fairli
None
Do you want to do stemming again? (yes/no): no
```



```
Select operation.
1.Porter Stemmer
2.Snowball Stemmer
3. Lancaster Stemmer
Enter choice (1/2/3): 2
Snowball Stemmer
run ----> run
running ----> run
ran ----> ran
runs ----> run
easily ----> easili
fairly ----> fair
None
Do you want to do stemming again? (yes/no): yes
Enter choice (1/2/3): 3
Lancaster Stemmer
run ----> run
running ----> run
ran ----> ran
runs ----> run
easily ----> easy
fairly ----> fair
None
Do you want to do stemming again? (yes/no): no
```

Name: Avinash R. Kauran, Roll No: CS22006
Course: Natural Language Processing

PRACTICAL NO : 03

AIM: Write a program to implement a Tri-Gram Model

THEORY:

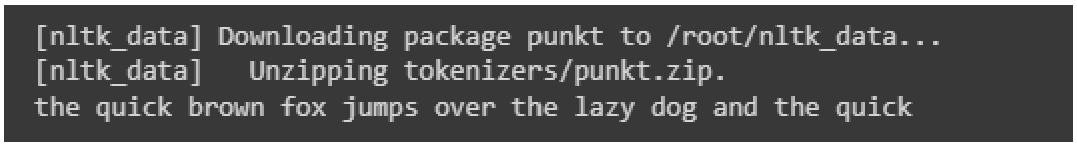
Tri-Gram Model

- ✓ A trigram model is a statistical language model used in natural language processing (NLP) that predicts the probability of the next word in a sequence,given the two previous words.
- ✓ It is based on the Markov assumption that the probability of a word dependsonly on the preceding two words, and not on any earlier context.
- ✓ The trigram model can be used in various NLP tasks such as language modeling,speech recognition, machine translation, and text classification.
- ✓ It is a simple yet effective model that can capture some of the syntactic andsemantic dependencies between words in a sentence.
- ✓ However, it suffers from the data sparsity problem, as the number of distinct trigrams in a large corpus can be very large, and many of them may not occur atall.
- ✓ Various techniques such as smoothing and backoff can be used to address thisproblem.

Source Code:

```
import nltk
from nltk import trigrams
from collections import Counter, defaultdict

# define the text corpus
corpus = "the quick brown fox jumps over the lazy dog and the quick brown fox jumps over the lazy dog again"

# tokenize the corpus into words
tokens = nltk.word_tokenize(corpus)

# create a trigram model
trigram_model = defaultdict(lambda: defaultdict(lambda: 0))
for w1, w2, w3 in trigrams(tokens, pad_right=True, pad_left=True):
    trigram_model[(w1, w2)][w3] += 1

# normalize the trigram frequencies
for w1_w2 in trigram_model:
    total_count = float(sum(trigram_model[w1_w2].values()))
    for w3 in trigram_model[w1_w2]:
        trigram_model[w1_w2][w3] /= total_count

# generate some sample text using the trigram model
text = ["the", "quick"]
for i in range(10):
    w1, w2 = text[-2], text[-1]
    w3 = max(trigram_model[(w1, w2)], key=trigram_model[(w1, w2)].get)
    text.append(w3)

# print the generated text
print(' '.join(text))
```

Output:



```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
the quick brown fox jumps over the lazy dog and the quick
```

Name: Avinash R. Kauran, Roll No: CS22006
Course: Natural Language Processing

PRACTICAL NO : 04

AIM: Write a program to Implement POS Tagging.

THEORY
What is POS Tagging?
Tagging is a kind of classification that may be defined as the automatic assignment of description to the tokens. Here the descriptor is called tag, which may represent one ofthe part-of-speech, semantic information and so on.

Part-of-Speech (PoS) tagging may be defined as the process of assigning one of the partsof speech to the given word. It is generally called POS tagging. In simple words, we can say that POS tagging is a task of labelling each word in a sentence with its appropriate part of speech. We already know that parts of speech include nouns, verb, adverbs, adjectives, pronouns, conjunction and their sub-categories.

Most of the POS tagging falls under Rule Base POS tagging, Stochastic POS tagging and Transformation based tagging.

Rule-based POS Tagging
One of the oldest techniques of tagging is rule-based POS tagging. Rule-based taggers use dictionary or lexicon for getting possible tags for tagging each word. If the word hasmore than one possible tag, then rule-based taggers use hand-written rules to identify the correct tag.

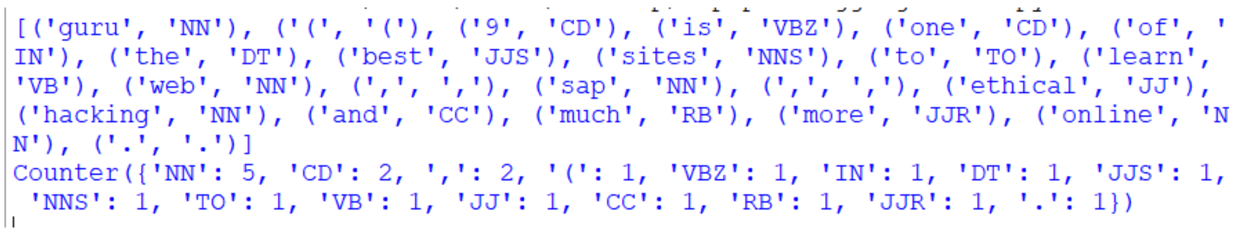
Stochastic POS Tagging
The model that includes frequency or probability (statistics) can be called stochastic. Any number of different approaches to the problem of part-of-speech tagging can be referred to as stochastic tagger.

Transformation-based Tagging
Transformation based tagging is also called Brill tagging. It is an instance of the transformation-based learning (TBL), which is a rule-based algorithm for automatictagging of POS to the given text. TBL, allows us to have linguistic knowledge in a readable form, transforms one state to another state by using transformation rules.

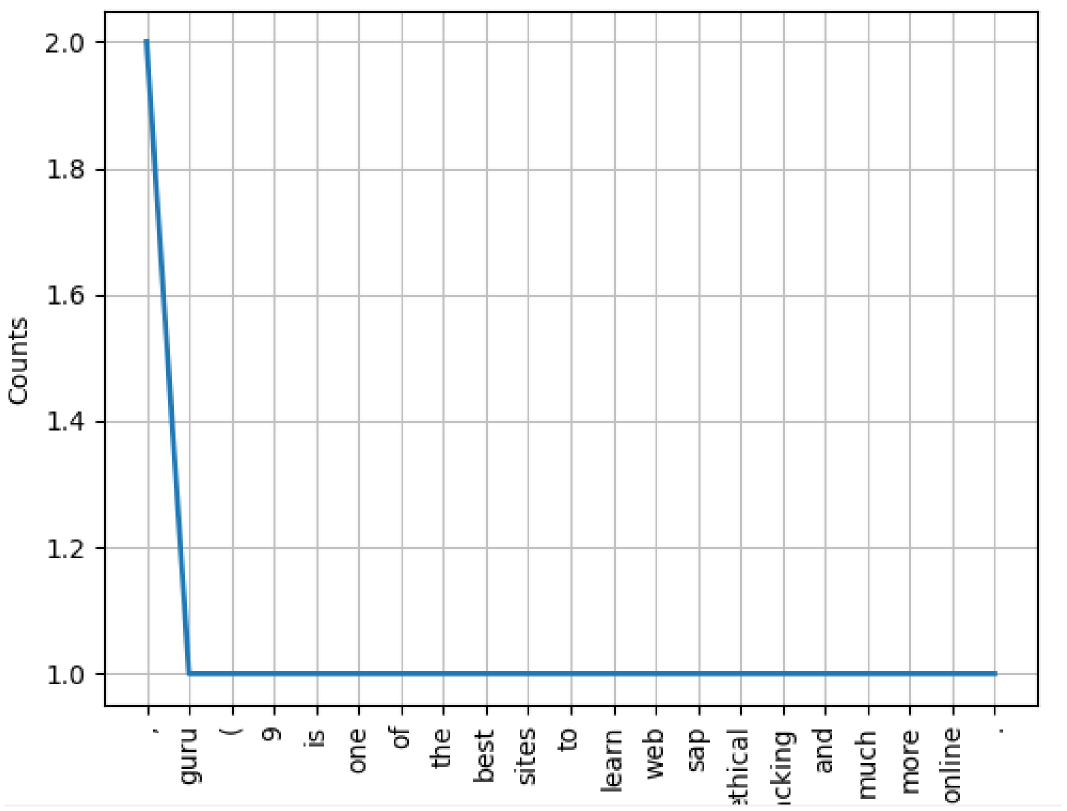
CODE:
```
import nltk
# nltk.download('punkt')
# nltk.download('averaged_perceptron_tagger')
from collections import Counter
text="Guru is one of the best sites to learn WEB,SAP,Ethical Hacking and much more online."
lower_case=text.lower()
tokens=nltk.word_tokenize(lower_case)
tags=nltk.pos_tag(tokens)
print(tags)
counts=Counter(tag for word,tag in tags)


###for tag in tags:
### print(tag)
print(counts)
fd=nltk.FreqDist(tokens)
fd.plot()
```

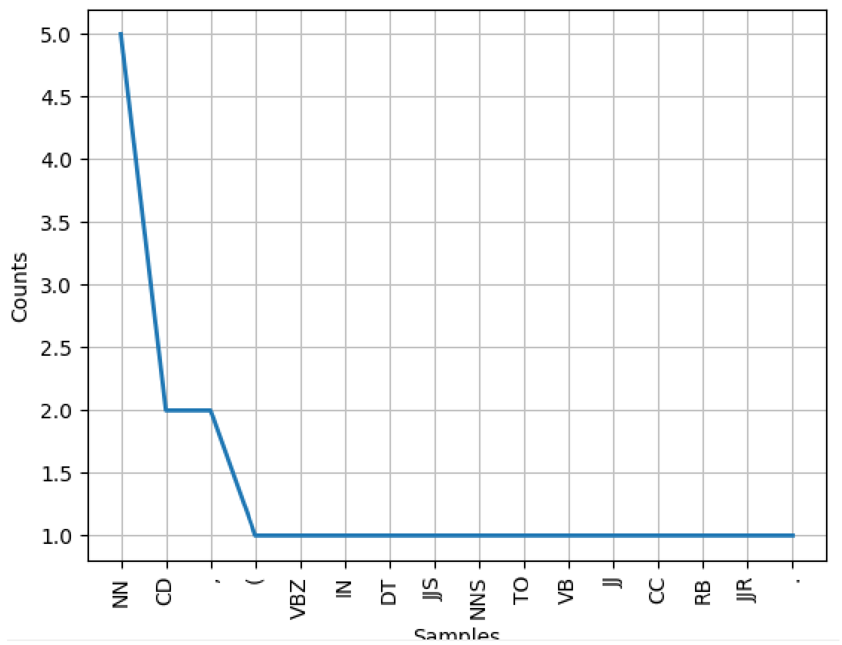
```
fd1=nltk.FreqDist(counts)
fd1.plot()
```

OUTPUT:



```
[('guru', 'NN'), ('(', '('), ('9', 'CD'), ('is', 'VBZ'), ('one', 'CD'), ('of', '
IN'), ('the', 'DT'), ('best', 'JJS'), ('sites', 'NNS'), ('to', 'TO'), ('learn',
'VB'), ('web', 'NN'), (',', ','), ('sap', 'NN'), (',', ','), ('ethical', 'JJ'),
('hacking', 'NN'), ('and', 'CC'), ('much', 'RB'), ('more', 'JJR'), ('online', 'N
N'), ('.', '.')]
Counter({'NN': 5, 'CD': 2, ',': 2, '(': 1, 'VBZ': 1, 'IN': 1, 'DT': 1, 'JJS': 1,
 'NNS': 1, 'TO': 1, 'VB': 1, 'JJ': 1, 'CC': 1, 'RB': 1, 'JJR': 1, '.': 1})
```





Name: Avinash R. Kauran, Roll No: CS22006
Course: Natural Language Processing

PRACTICAL NO : 05

AIM: Write a program to Implement Syntactic Parsing of a given text

THEORY:
What is Syntactic Parsing?
Syntactic analysis or parsing or syntax analysis is the third phase of NLP. The purpose ofthis phase is to draw exact meaning, or you can say dictionary meaning from the text. Syntax analysis checks the text for meaningfulness comparing to the rules of formal grammar.

Concept of Parser
It is used to implement the task of parsing. It may be defined as the software componentdesigned for taking input data (text) and giving structural representation of the input after checking for correct syntax as per formal grammar. It also builds a data structure generally in the form of parse tree or abstract syntax tree or other hierarchical structure.

.

```
#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))
# print(tagged)
#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(chunker)
# #print all parts of speech in above sentence.
output = chunker.parse(tagged)
print("After Extracting\n",output)
print(output)
```

O/P:

```
After Extracting
 (S
  (NP The/DT little/JJ yellow/JJ dog/NN)
  (VP (V barked/VBD) (PP (P at/IN) (NP the/DT cat/NN))))
(S
  (NP The/DT little/JJ yellow/JJ dog/NN)
  (VP (V barked/VBD) (PP (P at/IN) (NP the/DT cat/NN))))
```

Name: Avinash R. Kauran, Roll No: CS22006
Course: Natural Language Processing

PRACTICAL NO : 06

AIM: Write a program to Implement Dependency Parsing of a given text

THEORY:
Dependency Parsing
The term Dependency Parsing (DP) refers to the process of examining the dependenciesbetween the phrases of a sentence in order to determine its grammatical structure. A sentence is divided into many sections based mostly on this. The process is based on theassumption that there is a direct relationship between each linguistic unit in a sentence. These hyperlinks are called dependencies.

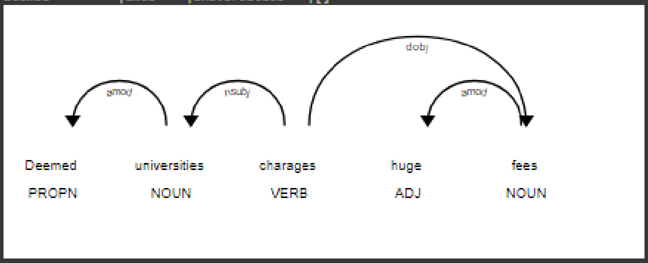
Module:
spaCy is a library for advanced Natural Language Processing in Python and Cython.

Code:

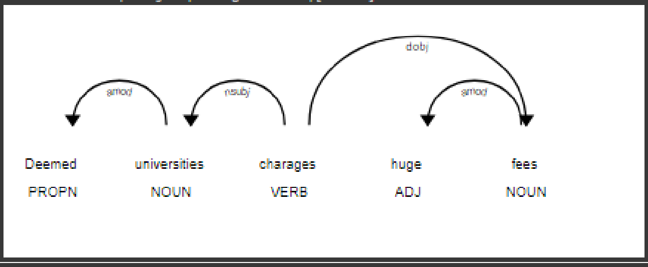
```
import spacy
from spacy import displacy

#load the language model
nlp = spacy.load("en_core_web_sm")
sentence = 'Deemed universities charages huge 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 child in
token.children])))
#use displayCy to visualize the dependencies
   displacy.render(doc,style='dep',jupyter=True,options={'distance':120})
```

O/P:

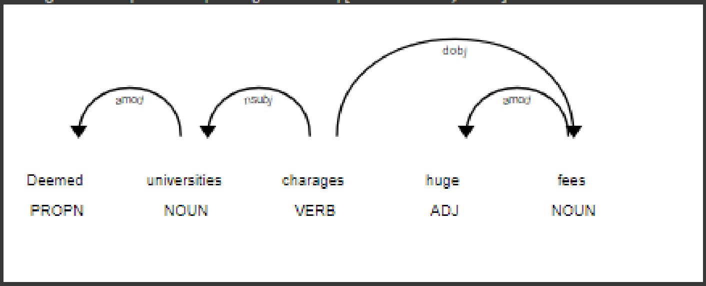
```
Token          |Relation|Head          |Children
---------------------------------------------------------------------------
Deemed         |amod    |universities  |[]
```



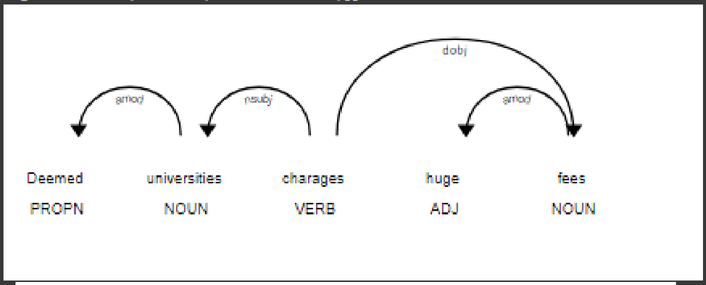
```
universities   |nsubj   |charages      |[Deemed]
```



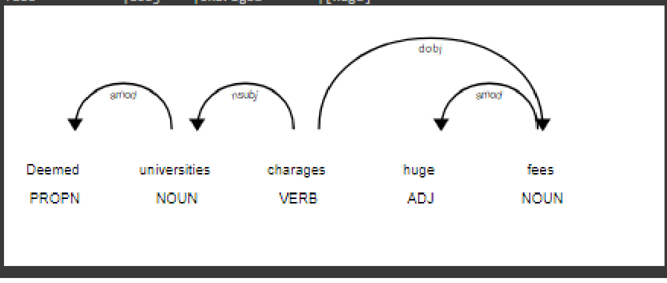
```
charages       |ROOT    |charages      |[universities, fees]
```



```
huge           |amod    |fees          |[]
```



```
fees           |dobj    |charages      |[huge]
```



Name: Avinash R. Kauran, Roll No: CS22006
Course: Natural Language Processing

PRACTICAL NO : 07

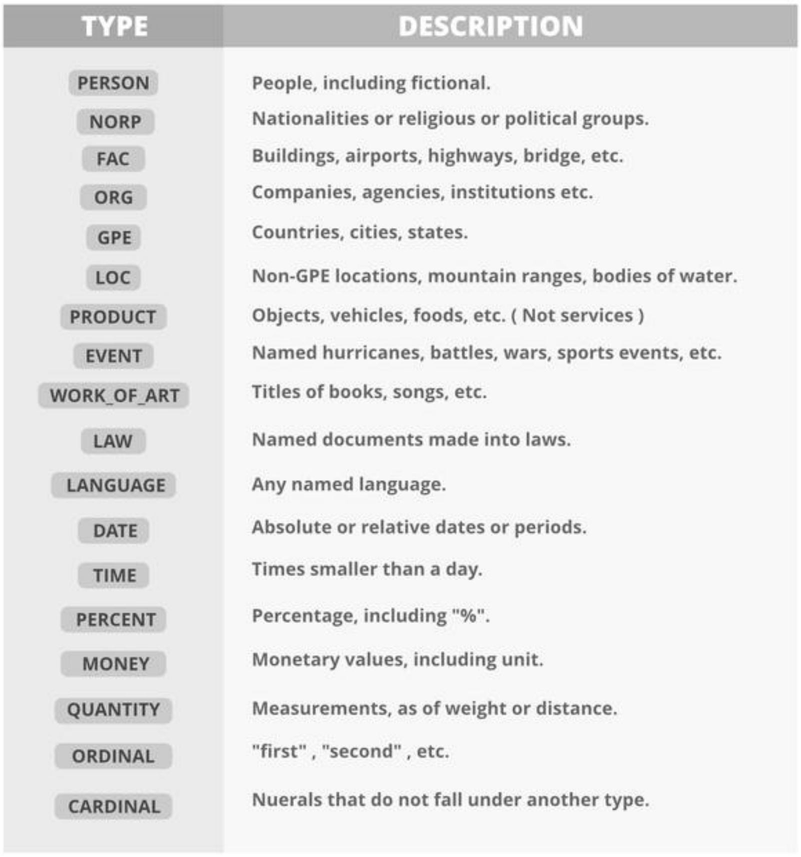
AIM: Write a program to implement Named Entity Recognition (NER)

THEORY:
What is NER?
Named-entity recognition is a subtask of information extraction that seeks to locate andclassify named entities mentioned in unstructured text into predefined categories such as person names, organizations, locations, medical codes, time expressions, quantities, monetary values, percentages, etc.

What is NER used for?
Named entity recognition (NER) helps you easily identify the key elements in a text, likenames of people, places, brands, monetary values, and more.

How do NER work?
Named Entity Recognition is a process where an algorithm takes a string of text (sentence or paragraph) as input and identifies relevant nouns (people, places, and organizations) that are mentioned in that string.

What is spacy used for?
spacy is a free, open-source library for NLP in Python. It's written in Cython and is designed to build information extraction or natural language understanding systems.

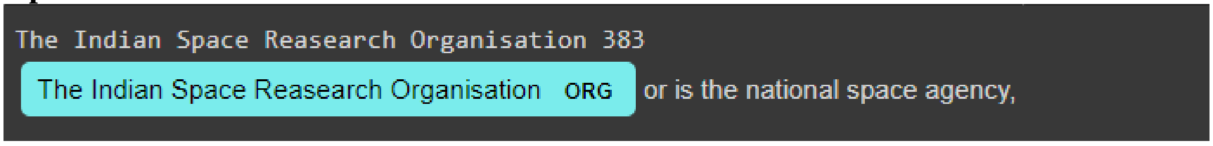


TYPE	DESCRIPTION
PERSON	People, including fictional.
NORP	Nationalities or religious or political groups.
FAC	Buildings, airports, highways, bridge, etc.
ORG	Companies, agencies, institutions etc.
GPE	Countries, cities, states.
LOC	Non-GPE locations, mountain ranges, bodies of water.
PRODUCT	Objects, vehicles, foods, etc. (Not services)
EVENT	Named hurricanes, battles, wars, sports events, etc.
WORK_OF_ART	Titles of books, songs, etc.
LAW	Named documents made into laws.
LANGUAGE	Any named language.
DATE	Absolute or relative dates or periods.
TIME	Times smaller than a day.
PERCENT	Percentage, including "%".
MONEY	Monetary values, including unit.
QUANTITY	Measurements, as of weight or distance.
ORDINAL	"first" , "second" , etc.
CARDINAL	Nuerals that do not fall under another type.

```
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)
```

Op



```
The Indian Space Reasearch Organisation 383
```

The Indian Space Reasearch Organisation ORG or is the national space agency,

Name: Avinash R. Kauran, Roll No: CS22006
Course: Natural Language Processing

PRACTICAL NO : 08

AIM: Write a program to Implement Text Summarization for the givensample text.

THEORY:
Text summarization is the process of creating a shorter version of a longer text while preserving its main points and essential meaning. The goal of text summarization is toreduce the amount of time and effort required to understand the content of a text, making it easier to digest and extract the key information.

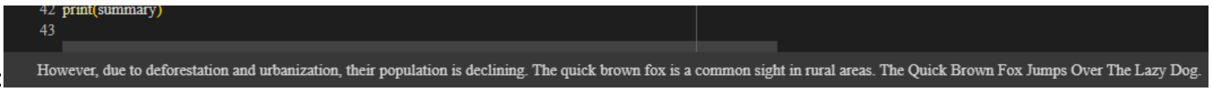
There are two main types of text summarization: extractive and abstractive. Extractivesummarization involves selecting and combining the most important sentences or phrases from the original text, while abstractive summarization involves generating new sentences that capture the meaning of the original text.

Text summarization is used in various fields, including news and media, research, and education. It can be done manually or with the help of automated tools and algorithms,such as natural language processing (NLP) techniques and machine learning models.

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import sent_tokenize, word_tokenize
from heapq import nlargest
nltk.download("punkt")
nltk.download('stopwords')

# define the text to be summarized
text = """The Quick Brown Fox Jumps Over The Lazy Dog. This is a simple example of text
summarization.
        The quick brown fox jumps over the lazy dog again. This is the second sentence.
        The quick brown fox is a common sight in rural areas. However, due to deforestation and
urbanization, their population is declining.
        Therefore, conservation efforts are necessary to protect them from extinction."""

# split the text into sentences
sentences = sent_tokenize(text)

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

# calculate the word frequency for each word in the text
word_frequencies = {}
for word in word_tokenize(text):
    if word.lower() not in stop_words:
        if word not in word_frequencies:
            word_frequencies[word] = 1
        else:
            word_frequencies[word] += 1

# calculate the sentence score based on the word frequency of each sentence
sentence_scores = {}
for sentence in sentences:
    for word in word_tokenize(sentence.lower()):
        if word in word_frequencies:
            if len(sentence.split(' ')) < 30:
```

```
            if sentence not in sentence_scores:
                sentence_scores[sentence] = word_frequencies[word]
            else:
                sentence_scores[sentence] += word_frequencies[word]

# get the top 3 sentences with the highest scores
summary_sentences = nlargest(3, sentence_scores, key=sentence_scores.get)

# print the summary
summary = ' '.join(summary_sentences)
print(summary)
```

OP:



```
print(summary)
```

```
However, due to deforestation and urbanization, their population is declining. The quick brown fox is a common sight in rural areas. The Quick Brown Fox Jumps Over The Lazy Dog.
```

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