**16. Implement a Python program using the SpaCy library to perform Named Entity Recognition (NER) on a given text.**

**Program:**

import spacy

def perform\_ner(text):

nlp = spacy.load("en\_core\_web\_sm")

doc = nlp(text)

for ent in doc.ents:

print(f"Entity: {ent.text}, Label: {ent.label\_}")

text = "Apple Inc. is an American multinational technology company headquartered in Cupertino, California."

perform\_ner(text)

**17. Write program demonstrates how to access WordNet, a lexical database, to retrieve synsets and explore word meanings in python.**

**Program:**

import nltk

# Download WordNet data

nltk.download('wordnet')

from nltk.corpus import wordnet

def explore\_word\_meanings(word):

synsets = wordnet.synsets(word)

if not synsets:

print(f"No synsets found for '{word}'.")

else:

print(f"Synsets for '{word}':")

for synset in synsets:

print(f" - {synset.name()} ({synset.pos()})")

print(f" Definition: {synset.definition()}")

print(f" Examples: {synset.examples()}")

print()

word\_to\_explore = "bottle"

explore\_word\_meanings(word\_to\_explore)

**18. Implement a simple FOPC parser for basic logical expressions using python program.**

**Program:**

from pyparsing import Word, alphanums, oneOf, infixNotation, opAssoc, Forward, ParseException

variable = Word(alphanums)

predicate = Word(alphanums)

quantifier = oneOf("forall exists")

expression = Forward()

unary\_operators = oneOf("not ¬")

quantifier\_operators = oneOf("forall exists")

atom = variable | predicate

expression << (

(quantifier + variable + "(" + expression + ")") |

(unary\_operators + expression) |

atom

)

def parse\_fopc(expression\_str):

try:

result = expression.parseString(expression\_str, parseAll=True)

return result[0]

except ParseException as e:

print(f"Error parsing expression: {e}")

return None

expression1 = "forall x (P(x) -> Q(x))"

expression2 = "exists y (P(y) & Q(y))"

parsed\_expression1 = parse\_fopc(expression1)

parsed\_expression2 = parse\_fopc(expression2)

print("Parsed Expression 1:", parsed\_expression1)

print("Parsed Expression 2:", parsed\_expression2)

**19. Create a program for word sense disambiguation using the Lesk algorithm using python.**

**Program:**

from nltk.corpus import wordnet

from nltk.tokenize import word\_tokenize

from nltk.corpus import stopwords

import nltk

nltk.download('punkt')

nltk.download('stopwords')

nltk.download('wordnet')

def lesk(word, sentence):

sentence\_tokens = set(word\_tokenize(sentence.lower()))

best\_sense = None

max\_overlap = 0

for synset in wordnet.synsets(word):

signature = set(word\_tokenize(synset.definition().lower()))

signature.update(word\_tokenize(" ".join(synset.examples()).lower()))

overlap = len(sentence\_tokens.intersection(signature))

if overlap > max\_overlap:

max\_overlap = overlap

best\_sense = synset

return best\_sense

word = "bank"

sentence = "He sat on the bank of the river and watched the sunset."

sense = lesk(word, sentence)

if sense:

print("Word:", word)

print("Sentence:", sentence)

print("Best Sense:", sense)

print("Definition:", sense.definition())

else:

print("No sense found for the word:", word)

**20. Implement a basic information retrieval system using TF-IDF (Term Frequency-Inverse Document Frequency) for document ranking using python.**

**Program:**

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import linear\_kernel

documents = [

"TF-IDF, or term frequency-inverse document frequency, is a statistic that evaluates the importance of a word in a document relative to a collection of documents.",

"It is often used in information retrieval and text mining.",

"The TF-IDF score increases with the frequency of the word in the document and is offset by the frequency of the word in the entire collection of documents.",

"In other words, TF-IDF highlights words that are unique to a document and are not common across many documents.",

"To implement TF-IDF, you need to calculate the term frequency and inverse document frequency for each word in the document collection.",

]

query = "TF-IDF in information retrieval"

vectorizer = TfidfVectorizer()

tfidf\_matrix = vectorizer.fit\_transform(documents + [query])

cosine\_similarities = linear\_kernel(tfidf\_matrix[-1], tfidf\_matrix[:-1]).flatten()

document\_ranking = sorted(enumerate(cosine\_similarities), key=lambda x: x[1], reverse=True)

print("Ranked Documents:")

for idx, similarity in document\_ranking:

print(f"Document {idx + 1}: Similarity = {similarity:.4f}")

print(f" {documents[idx]}")

print()