**21. Create a python program that performs syntax-driven semantic analysis by extracting noun phrases and their meanings from a sentence.**

**Program:**

import nltk

from nltk import pos\_tag

from nltk.tokenize import word\_tokenize

from nltk.chunk import ne\_chunk

# Download NLTK resources if not already downloaded

nltk.download('punkt')

nltk.download('averaged\_perceptron\_tagger')

nltk.download('maxent\_ne\_chunker')

nltk.download('words')

def extract\_noun\_phrases(sentence):

words = word\_tokenize(sentence)

tagged\_words = pos\_tag(words)

chunked\_sentence = ne\_chunk(tagged\_words)

noun\_phrases = []

for subtree in chunked\_sentence:

if isinstance(subtree, nltk.tree.Tree) and subtree.label() == 'NP':

noun\_phrase = " ".join(word for word, tag in subtree.leaves())

noun\_phrases.append(noun\_phrase)

return noun\_phrases

def determine\_meaning(noun\_phrase):

# Dummy implementation, replace this with your actual logic to determine meanings

# For this example, we'll just return a placeholder meaning

meanings = {

"cat": "a small domesticated carnivorous mammal with soft fur, a short snout, and retractile claws",

"ball": "a round or spherical object that is used in games and sports"

}

return meanings.get(noun\_phrase, "Meaning not found")

def main():

sentence = input("Enter a sentence: ")

noun\_phrases = extract\_noun\_phrases(sentence)

print("\nNoun phrases found in the sentence:")

for noun\_phrase in noun\_phrases:

print("- {}".format(noun\_phrase))

meaning = determine\_meaning(noun\_phrase.lower())

print(" Meaning:", meaning)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**22. Create a python program that performs reference resolution within a text.**

**Program:**

import nltk

from nltk.tokenize import sent\_tokenize, word\_tokenize

from nltk.tag import pos\_tag

from nltk.chunk import ne\_chunk

# Download NLTK resources if not already downloaded

nltk.download('punkt')

nltk.download('averaged\_perceptron\_tagger')

nltk.download('maxent\_ne\_chunker')

nltk.download('words')

def resolve\_references(text):

sentences = sent\_tokenize(text)

resolved\_text = ""

for sentence in sentences:

tagged\_sentence = pos\_tag(word\_tokenize(sentence))

chunked\_sentence = ne\_chunk(tagged\_sentence)

resolved\_sentence = ""

for subtree in chunked\_sentence:

if type(subtree) == nltk.tree.Tree:

if subtree.label() == 'NP': # Noun phrase

antecedent = find\_antecedent(subtree)

resolved\_sentence += antecedent + " "

else:

resolved\_sentence += " ".join(word for word, tag in subtree.leaves()) + " "

else:

resolved\_sentence += subtree[0] + " "

resolved\_text += resolved\_sentence.strip() + "\n"

return resolved\_text

def find\_antecedent(subtree):

for node in subtree:

if type(node) == nltk.tree.Tree:

if node.label() == 'NP':

return find\_antecedent(node)

else:

if node[1] == 'PRP':

antecedent = get\_antecedent(node)

return antecedent

return ""

def get\_antecedent(pronoun):

antecedent = ""

if pronoun[0].lower() == 'he':

antecedent = "John" # Replace with actual antecedent from context

elif pronoun[0].lower() == 'she':

antecedent = "Mary" # Replace with actual antecedent from context

# Add more cases for other pronouns as needed

return antecedent

def main():

text = """

John saw Mary at the store. She was buying groceries.

He decided to greet her.

"""

resolved\_text = resolve\_references(text)

print("Original text:")

print(text)

print("\nResolved text:")

print(resolved\_text)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**23. Develop a python program that evaluates the coherence of a given text.**

**Program:**

import nltk

from nltk.tokenize import sent\_tokenize, word\_tokenize

from nltk.corpus import stopwords

from nltk.collocations import BigramAssocMeasures, BigramCollocationFinder

# Download NLTK resources if not already downloaded

nltk.download('punkt')

nltk.download('stopwords')

def calculate\_coherence(text):

sentences = sent\_tokenize(text)

word\_tokens = [word.lower() for sentence in sentences for word in word\_tokenize(sentence) if word.isalnum()]

filtered\_tokens = [word for word in word\_tokens if word not in stopwords.words('english')]

bigram\_measures = BigramAssocMeasures()

finder = BigramCollocationFinder.from\_words(filtered\_tokens)

pmi = finder.score\_ngrams(bigram\_measures.pmi)

coherence\_score = sum(score for bigram, score in pmi) / len(pmi)

return coherence\_score

def main():

text = """

Coherence in writing means that all the ideas in a paragraph flow smoothly from one sentence to the next.

With each sentence building on the previous one and leading logically to the next.

Achieving coherence requires paying attention to the organization and development of ideas.

"""

coherence\_score = calculate\_coherence(text)

print("Coherence score:", coherence\_score)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**24. Create a python program that recognizes dialog acts in a given dialog or conversation.**

**Program:**

import nltk

from nltk.tokenize import sent\_tokenize, word\_tokenize

# Download NLTK resources if not already downloaded

nltk.download('punkt')

def recognize\_dialog\_acts(dialog):

sentences = sent\_tokenize(dialog)

dialog\_acts = []

for sentence in sentences:

tokens = word\_tokenize(sentence.lower())

dialog\_act = classify\_dialog\_act(tokens)

dialog\_acts.append((sentence, dialog\_act))

return dialog\_acts

def classify\_dialog\_act(tokens):

greetings = ['hello', 'hi', 'hey']

farewells = ['goodbye', 'bye', 'see you']

questions = ['what', 'where', 'when', 'who', 'why', 'how']

statements = ['i think', 'i believe', 'in my opinion']

if any(token in greetings for token in tokens):

return 'GREETING'

elif any(token in farewells for token in tokens):

return 'FAREWELL'

elif any(token in questions for token in tokens):

return 'QUESTION'

elif any(token in statements for token in tokens):

return 'STATEMENT'

else:

return 'OTHER'

def main():

dialog = """

Speaker 1: Hi, how are you?

Speaker 2: I'm doing well, thank you. How about you?

Speaker 1: Goodbye then, have a great day!

"""

dialog\_acts = recognize\_dialog\_acts(dialog)

print("Dialog Acts:")

for utterance, dialog\_act in dialog\_acts:

print(f"{utterance.strip()} - {dialog\_act}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**25. Utilize the GPT-3 model to generate text based on a given prompt. Make sure to install the OpenAI GPT-3 library in python implementation.**

**Program:**

import openai

# Set your OpenAI API key

api\_key = 'YOUR\_API\_KEY'

openai.api\_key = api\_key

def generate\_text(prompt, max\_tokens=50):

response = openai.Completion.create(

engine="text-davinci-002", # Choose the GPT-3 engine you prefer

prompt=prompt,

max\_tokens=max\_tokens

)

return response.choices[0].text.strip()

def main():

prompt = "Once upon a time"

generated\_text = generate\_text(prompt)

print("Generated text:")

print(generated\_text)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**26. Implement a machine translation program using the Hugging Face Transformers library,  translate English text to French using python**

**Program:**

from transformers import MarianMTModel, MarianTokenizer

def translate\_text(text, model\_name="Helsinki-NLP/opus-mt-en-fr"):

# Load pre-trained model and tokenizer

model = MarianMTModel.from\_pretrained(model\_name)

tokenizer = MarianTokenizer.from\_pretrained(model\_name)

# Tokenize input text

inputs = tokenizer(text, return\_tensors="pt", truncation=True, padding=True)

# Perform translation

outputs = model.generate(\*\*inputs)

# Decode the translated text

translated\_text = tokenizer.decode(outputs[0], skip\_special\_tokens=True)

return translated\_text

def main():

# Input English text

english\_text = "Hello, how are you?"

# Translate English text to French

french\_text = translate\_text(english\_text)

# Output translated text

print("Translated French text:")

print(french\_text)

if \_\_name\_\_ == "\_\_main\_\_":

main()