**Practical No. 1**

**Sentence Segmentation and Word Tokenization**

**Aim:** To split a paragraph into sentences and words using NLP libraries.

**Solution:**

**# Code: Using NLTK**

import nltk

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

text = "Natural Language Processing is fascinating. It helps computers understand human language. Let's explore NLP!"

sentences\_nltk = sent\_tokenize(text)

print("--- NLTK Output ---")

print("Sentences (NLTK):")

for i, sentence in enumerate(sentences\_nltk, 1):

print(f"{i}. {sentence}")

words\_nltk = word\_tokenize(text)

print("\nWords (NLTK):")

print(words\_nltk)

print()

**# Code: Using spaCy**

import spacy

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

doc = nlp(text)

print("\n--- spaCy Output ---")

print("Sentences (spaCy):")

for i, sent in enumerate(doc.sents, 1):

print(f"{i}. {sent.text}")

print("\nWords (spaCy):")

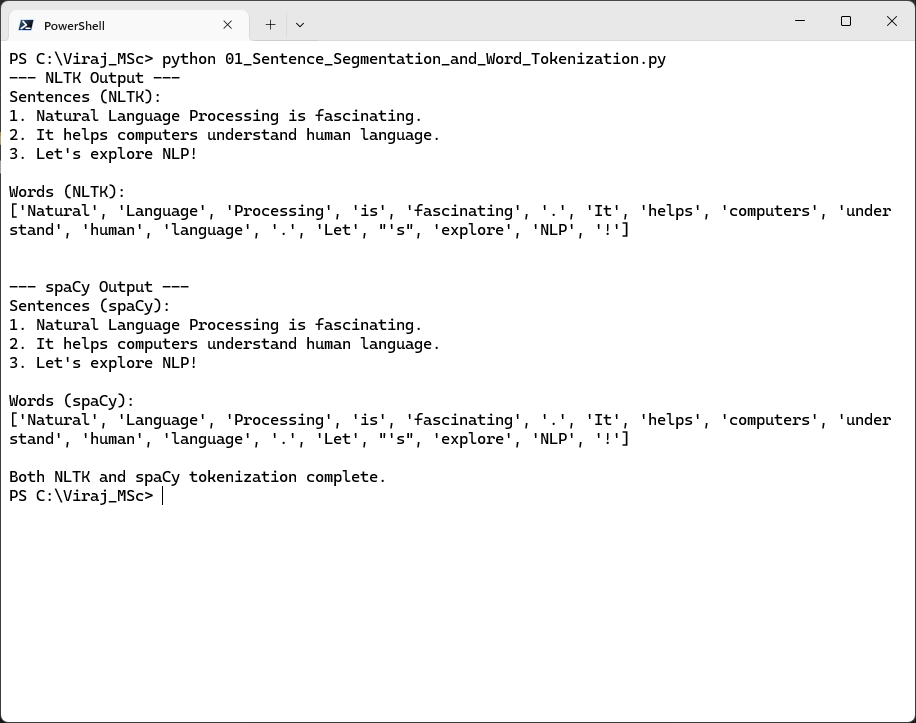
words\_spacy = [token.text for token in doc]

print(words\_spacy)

print("\nBoth NLTK and spaCy tokenization complete.")

**Observations and Results:**

* NLTK and spaCy both successfully segmented sentences and tokenized words.
* spaCy provides more linguistic features (e.g., dependency parsing) out of the box.

**Output:**

**Practical No. 2**

**Stemming and Lemmatization**

**Aim:** To reduce words to their root forms using stemming and lemmatization.

**Solution:**

**# Code: Using NLTK**

import nltk

from nltk.stem import PorterStemmer, WordNetLemmatizer

words = ["running", "flies", "happily", "better", "cats", "dogs", "ran"]

print("--- NLTK Output (Stemming & Lemmatization) ---")

# Stemming

stemmer = PorterStemmer()

stemmed = [stemmer.stem(word) for word in words]

print("Original Words:", words)

print("Stemmed (Porter):", stemmed)

# Lemmatization

lemmatizer = WordNetLemmatizer()

lemmatized\_verbs = [lemmatizer.lemmatize(word, pos='v') for word in words]

lemmatized\_nouns\_default = [lemmatizer.lemmatize(word) for word in words] # default pos is 'n'

print("Lemmatized (WordNet, pos='v'):", lemmatized\_verbs)

print("Lemmatized (WordNet, default pos='n'):", lemmatized\_nouns\_default)

**# Code: Using spaCy**

import spacy

spacy\_text = "running flies happily better cats dogs ran"

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

doc = nlp(spacy\_text)

print("\n--- spaCy Output (Lemmatization) ---")

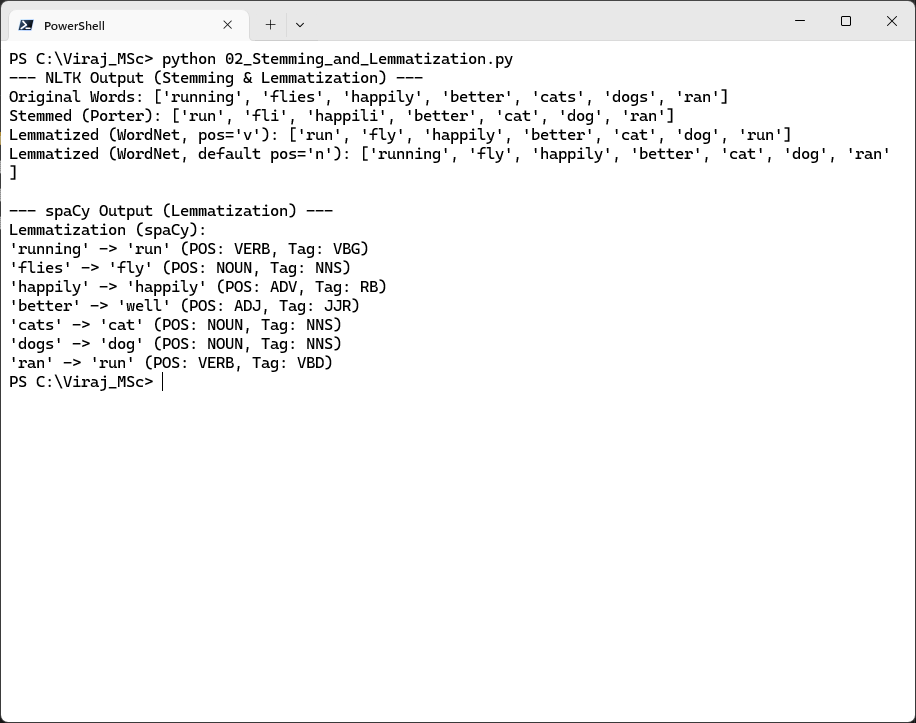
print("Lemmatization (spaCy):")

for token in doc:

print(f"'{token.text}' -> '{token.lemma\_}' (POS: {token.pos\_}, Tag: {token.tag\_})")

**Observations and Results:**

* Stemming (Porter) aggressively chops words (e.g., "flies" → "fly").
* Lemmatization (WordNet/spaCy) provides meaningful base forms (e.g., "dogs" → "dog").

**Output:**

**Practical No. 3**

**Implement a Tri-gram Model**

**Aim:** To generate tri-grams from a given text corpus.

**Solution:**

**# Code: Using NLTK**

import nltk

from nltk.util import ngrams

from nltk.tokenize import word\_tokenize

text = "Natural Language Processing is fun and challenging."

tokens = word\_tokenize(text)

print("--- NLTK Fixed Tri-grams ---")

print(f"Original Text: \"{text}\"")

print(f"NLTK Tokens: {tokens}")

# Generate tri-grams

tri\_grams\_nltk = list(ngrams(tokens, 3))

print("Tri-grams (NLTK - fixed sequence):")

for gram in tri\_grams\_nltk:

print(gram)

**# Code: Using spaCy**

import spacy

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

doc = nlp(text)

print("\n--- spaCy Tri-gram Extraction ---")

print(f"Original Text: \"{text}\"")

print(f"spaCy Tokens: {[token.text for token in doc]}")

tri\_grams\_spacy = []

for i in range(len(doc) - 2):

trigram = doc[i:i+3].text # .text joins the tokens into a string

tri\_grams\_spacy.append(trigram)

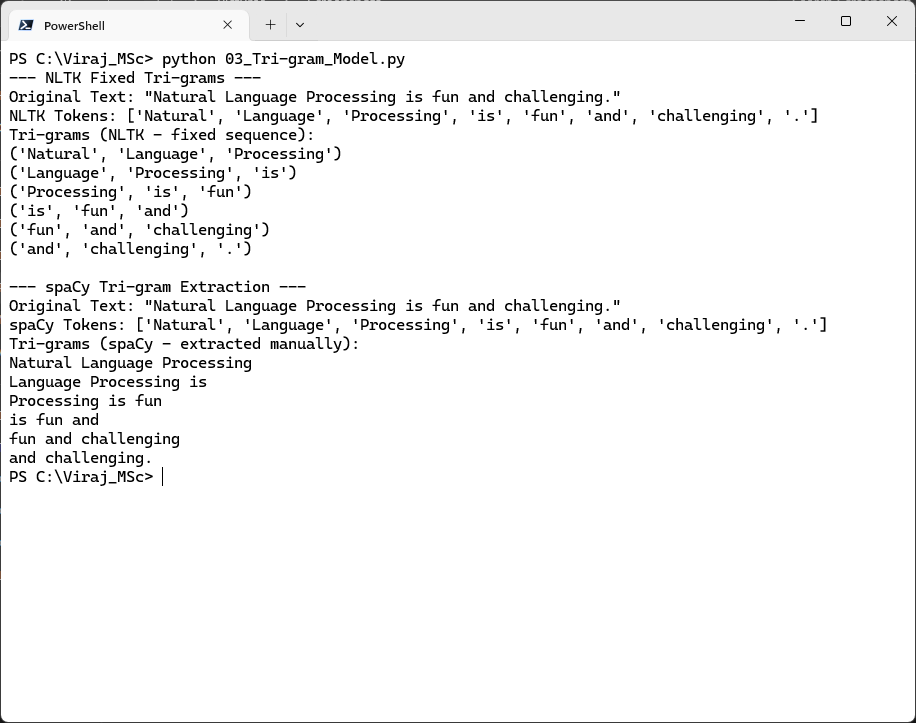
print("Tri-grams (spaCy - extracted manually):")

for gram in tri\_grams\_spacy:

print(gram)

**Observations and Results:**

* NLTK extracts fixed tri-grams (e.g., ('Natural', 'Language', 'Processing')).
* Gensim learns meaningful collocations (e.g., "natural\_language\_processing").

**Output:**

**Practical No. 4**

**PoS Tagging using HMM & Neural Model**

**Aim:** To tag parts of speech using Hidden Markov Models (HMM) and neural models.

**Solution:**

**# Code: Using NLTK**

import nltk

from nltk.tokenize import word\_tokenize # Import word\_tokenize

text = "I love Natural Language Processing. It is a fascinating field."

tokens = word\_tokenize(text)

print("--- NLTK (HMM-based) PoS Tags ---")

print(f"Text: \"{text}\"")

print(f"Tokens: {tokens}")

pos\_tags\_nltk = nltk.pos\_tag(tokens)

print("NLTK PoS Tags:")

print(pos\_tags\_nltk)

**# Code: Using spaCy**

import spacy

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

doc = nlp(text) # Use the same text as NLTK for comparison

print("\n--- spaCy (Neural Model) PoS Tags ---")

print(f"Text: \"{text}\"")

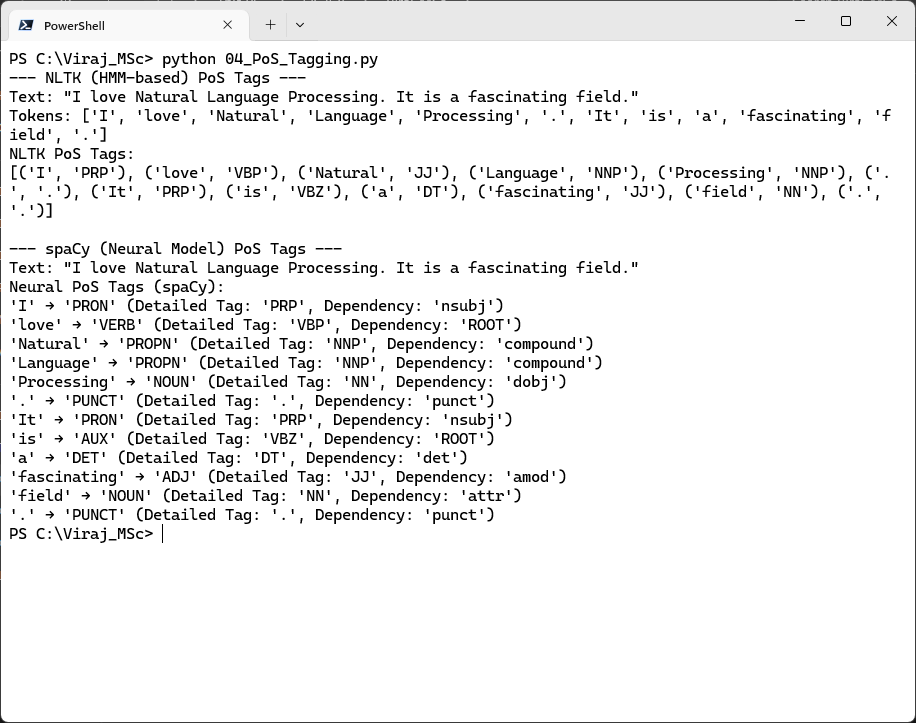
print("Neural PoS Tags (spaCy):")

for token in doc:

print(f"'{token.text}' → '{token.pos\_}' (Detailed Tag: '{token.tag\_}', Dependency: '{token.dep\_}')")

**Observations and Results:**

* HMM (NLTK): Tags like ('love', 'VBP') (verb).
* Neural (spaCy): More accurate with context (e.g., "Processing" → PROPN).

**Output:**

**Practical No. 5**

**Syntactic Parsing**

**Aim:** To parse sentence structure using constituency parsing.

**Solution:**

**# Code: Using NLTK**

import nltk

from nltk.tokenize import word\_tokenize

grammar = nltk.CFG.fromstring("""

S -> NP VP

VP -> V NP

NP -> Det N | 'I'

Det -> 'the'

N -> 'dog' | 'cat'

V -> 'chased'

""")

parser = nltk.ChartParser(grammar)

sentence\_nltk = "I chased the cat".split()

print("--- NLTK Constituency Parse Trees ---")

print(f"Sentence: \"{' '.join(sentence\_nltk)}\"")

print("Syntactic Parse Trees (NLTK):")

parsed\_trees\_count = 0

for tree in parser.parse(sentence\_nltk):

tree.pretty\_print()

parsed\_trees\_count += 1

if parsed\_trees\_count == 0:

print("No parse trees found for this sentence with the given grammar.")

**# Code: Using spaCy**

import os

import webbrowser

import spacy

from spacy import displacy

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

text\_spacy = "The dog chased the cat." # Sentence for spaCy

doc = nlp(text\_spacy)

print("\n--- spaCy Dependency Parsing (Textual Output) ---")

print(f"Sentence: \"{text\_spacy}\"")

print("Neural Dependency Tags (spaCy - Head --(Dependency)--> Dependent):")

for token in doc:

print(f"'{token.text}' --({token.dep\_})--> '{token.head.text}' [POS: {token.pos\_}, Detailed Tag: {token.tag\_}]")

# --- spaCy Visualization (Save to File & Open) ---

html\_output = displacy.render(doc, style="dep", jupyter=False)

script\_dir = os.path.dirname(\_\_file\_\_) # Get the directory of the current script

file\_name = "05\_Syntactic\_Parsing\_screenshot.html"

output\_file\_path = os.path.join(script\_dir, file\_name)

with open(output\_file\_path, "w", encoding="utf-8") as f:

f.write(html\_output)

print(f"\nSpaCy visualization saved to: {os.path.abspath(output\_file\_path)}")

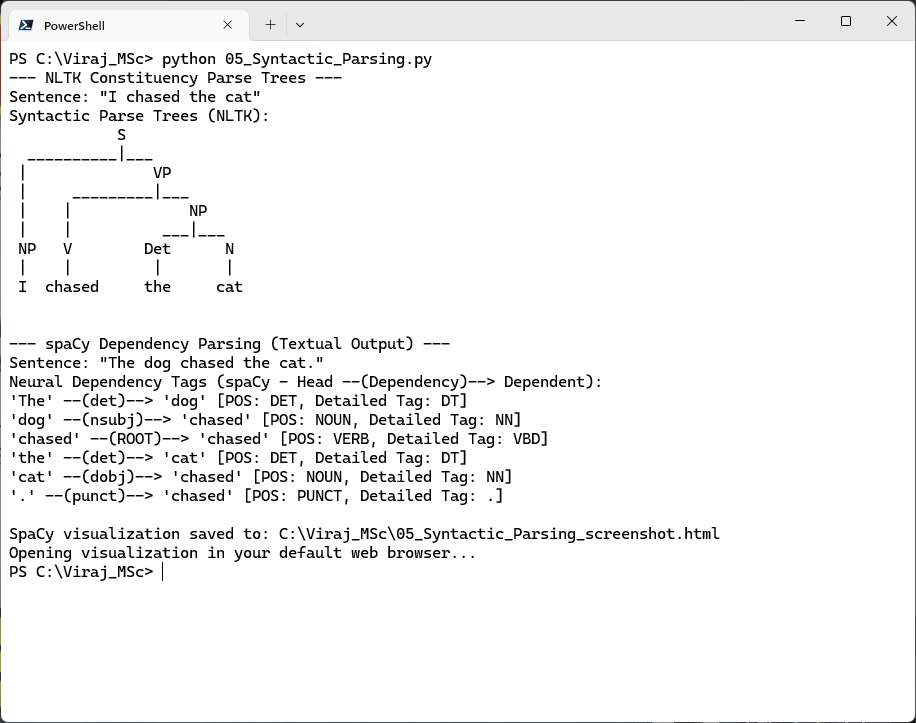
print("Opening visualization in your default web browser...")

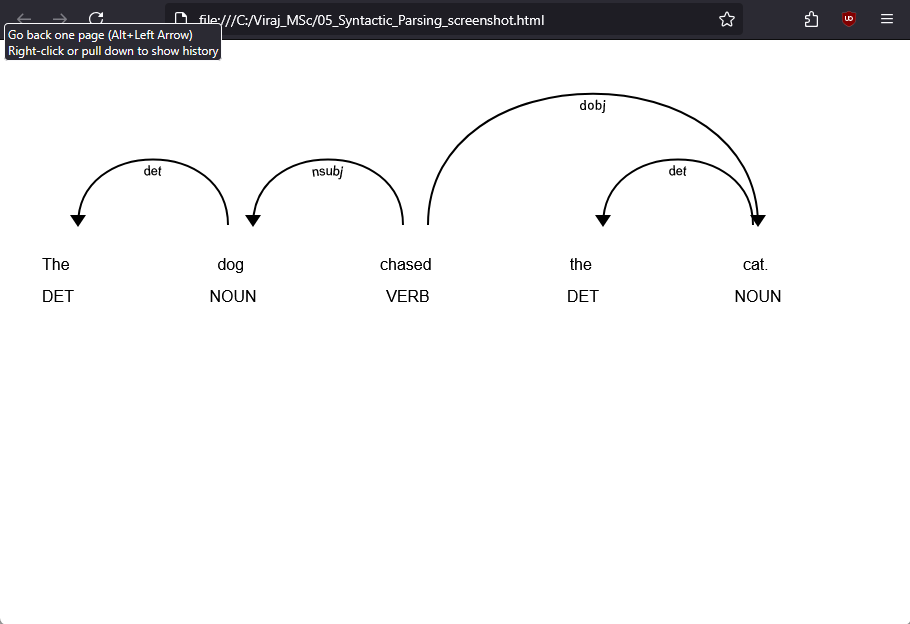
# Open the file in the default web browser

webbrowser.open\_new\_tab("file://" + os.path.abspath(output\_file\_path))

**Observations and Results:**

* NLTK generates constituency parse trees (rule-based).
* spaCy provides interactive dependency visualization.

**Output:**



**Practical No. 6**

**Dependency Parsing**

**Aim:** To analyze grammatical dependencies between words.

**Solution:**

**# Code: Using spaCy**

import spacy

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

text = "Natural Language Processing is fascinating."

doc = nlp(text)

print("--- spaCy Dependency Parsing ---")

print(f"Sentence: \"{text}\"")

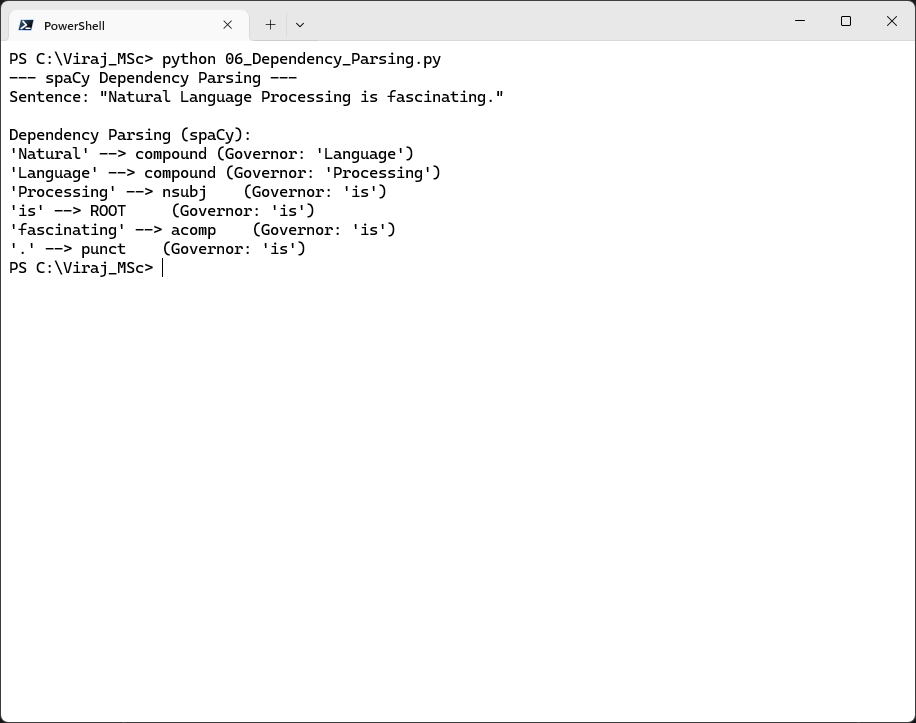
print("\nDependency Parsing (spaCy):")

for token in doc:

print(f"'{token.text}' --> {token.dep\_:<8} (Governor: '{token.head.text}')")

**Observations and Results:**

* spaCy accurately identifies subject-verb-object relationships.
* Example Output:
* Processing → nsubj (Governor: is)
* fascinating → acomp (Governor: is)

**Output:**