# INDEX

|  |  |  |
| --- | --- | --- |
| SrNo | Topics | Sign |
| 1 | Write a program to implement sentence segmentation and word tokenization |  |
| 2 | Write a program to Implement stemming and lemmatization |  |
| 3 | Write a program to Implement syntactic parsing of a given text. |  |
| 4 | Write a program to Implement dependency parsing of a given text |  |
| 5 | Write a program to Implement Named Entity Recognition (NER). |  |

# 1.Pratical no:

**1. Write a program to implement sentence segmentation and word tokenization. Input:**

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

# Download the necessary resources nltk.download('punkt')

def segment\_sentences(text):

"""

Segment the input text into sentences.

:param text: A string containing the text to be segmented.

:return: A list of sentences.

"""

sentences = sent\_tokenize(text) return sentences

def tokenize\_words(sentences): """

Tokenize the input sentences into words.

:param sentences: A list of sentences.

:return: A list of lists, where each inner list contains the words of the corresponding sentence. """

word\_tokens = [word\_tokenize(sentence) for sentence in sentences] return word\_tokens

if \_\_name\_\_ == "\_\_main\_\_": text = "Hello world! This is a test sentence. Sentence segmentation and word

tokenization are important preprocessing steps."

# Segment the text into sentences sentences = segment\_sentences(text) print("Sentences:") for i, sentence in enumerate(sentences): print(f"{i+1}: {sentence}")

# Tokenize each sentence into words word\_tokens = tokenize\_words(sentences) print("\nWord Tokens:")

for i, words in enumerate(word\_tokens): print(f"Sentence {i+1} words: {words}")

**Output:**



# Pratical no: 02

**2. Write a program to implement stemming and lemmatization. Input:**

import nltk

from nltk.stem import PorterStemmer, WordNetLemmatizer from nltk.tokenize import word\_tokenize

# Download the necessary resources

nltk.download('punkt') nltk.download('wordnet') nltk.download('omw-1.4')

def perform\_stemming(words): """

Perform stemming on the input words.

:param words: A list of words to be stemmed.

:return: A list of stemmed words.

"""

stemmer = PorterStemmer()

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

def perform\_lemmatization(words): """

Perform lemmatization on the input words.

:param words: A list of words to be lemmatized.

:return: A list of lemmatized words.

"""

lemmatizer = WordNetLemmatizer()

lemmatized\_words = [lemmatizer.lemmatize(word) for word in words] return lemmatized\_words

if \_\_name\_\_ == "\_\_main\_\_": text = "The striped bats are hanging on their feet for best"

# Tokenize the text into words words = word\_tokenize(text) print("Original Words:")

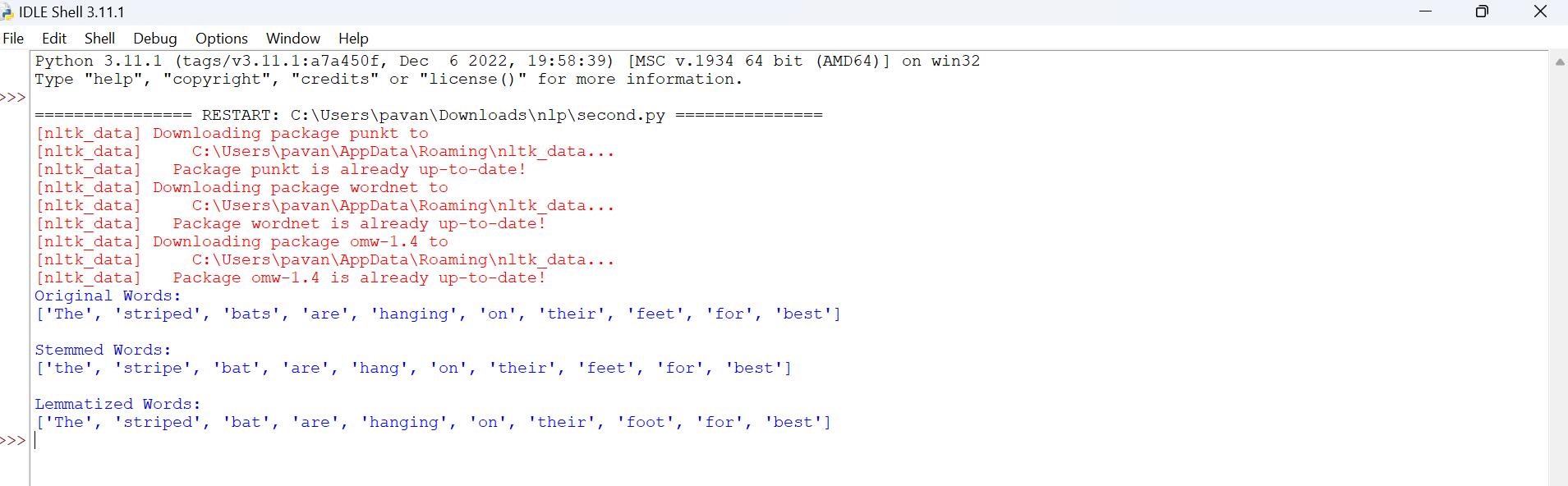
print(words)

# Perform stemming

stemmed\_words = perform\_stemming(words) print("\nStemmed Words:") print(stemmed\_words)

# Perform lemmatization lemmatized\_words = perform\_lemmatization(words) print("\nLemmatized Words:") print(lemmatized\_words)

**Output:**



# Pratical:03

**3.Write a program to Implement syntactic parsing of a given text.**

**Input:**

import nltk from nltk import CFG from nltk.parse.generate import generate

# Define a simple grammar

grammar = CFG.fromstring("""

S -> NP VP

VP -> V NP | V NP PP

PP -> P NP

V -> "saw" | "ate" | "walked"

NP -> "John" | "Mary" | "Bob" | Det N | Det N PP

Det -> "a" | "an" | "the" | "my"

N -> "man" | "dog" | "cat" | "telescope" | "park"

P -> "in" | "on" | "by" | "with" """)

# Create a parser parser = nltk.ChartParser(grammar)

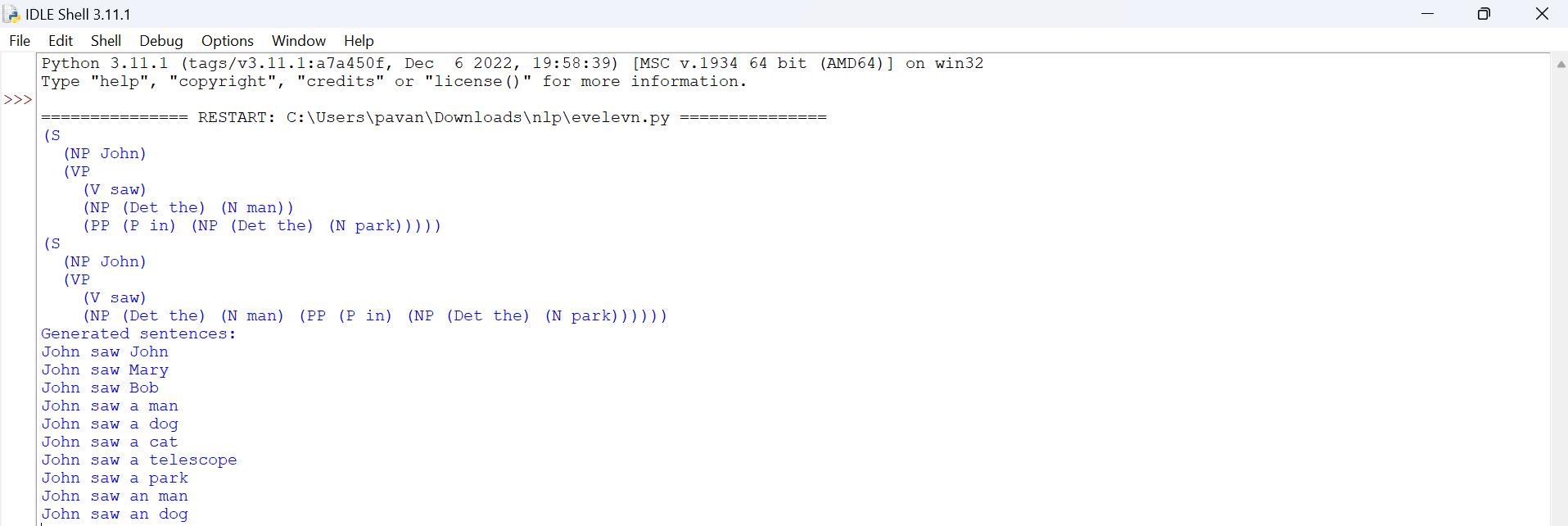
# Define a test sentence sentence = "John saw the man in the park".split()

# Parse the sentence parses = list(parser.parse(sentence))

# Display the parse trees for tree in parses: print(tree) tree.draw()

# If you want to generate all possible sentences according to the grammar print("Generated sentences:") for sentence in generate(grammar, n=10): print(' '.join(sentence))

**Output**:



# Pratical no:04

**4.Write a program to Implement dependency parsing of a given text. Input:**

import spacy

# Load the pre-trained spaCy model nlp = spacy.load("en\_core\_web\_sm")

# Define a test sentence sentence = "John saw the man in the park."

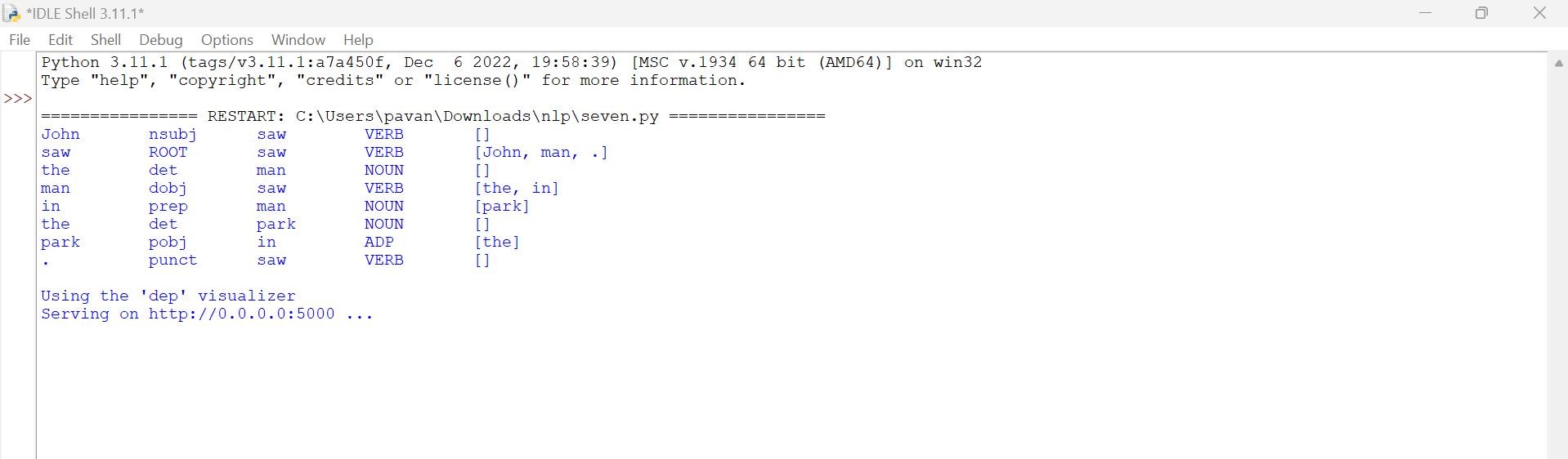
# Parse the sentence doc = nlp(sentence)

# Display the syntactic structure for token in doc: print(f"{token.text:10} {token.dep\_:10} {token.head.text:10} {token.head.pos\_:10}

{[child for child in token.children]}")

# Visualize the parse tree spacy.displacy.serve(doc, style="dep")

**Output:**



# Practical:05

**5.Write a program to Implement Named Entity Recognition (NER).**

**Input:**

import spacy

# Load the pre-trained spaCy model nlp = spacy.load("en\_core\_web\_sm")

# Define a test sentence

text = "Apple is looking at buying U.K. startup for $1 billion. Barack Obama was born on August 4, 1961."

# Process the text doc = nlp(text)

# Display the named entities print("Named Entities, their labels, and explanations:") for ent in doc.ents: print(f"{ent.text:20} {ent.label\_:10} {spacy.explain(ent.label\_)}")

# Visualize the named entities spacy.displacy.serve(doc, style="ent")

Input

