1 and 2 same but rules are different: check rules

import nltk

from nltk import CFG

grammar = CFG.fromstring("""

S -> NP VP

NP -> Det Nom

VP -> V NP

Nom -> Adj Nom | N

Det -> 'the'

Adj -> 'little' | 'angry' | 'frightened'

N -> 'squirrel' | 'bear'

V -> 'chased'

""")

sentence = ['the', 'angry', 'bear', 'chased', 'the', 'frightened', 'little', 'squirrel']

parser = nltk.ChartParser(grammar)

for tree in parser.parse(sentence):

tree.pretty\_print()

break

4.

import nltk

from nltk.util import ngrams

from collections import Counter

from nltk.tokenize import word\_tokenize

nltk.download('punkt')

corpus = """

<s> Do I like apples </s>

<s> Do I like bananas </s>

<s> Do you like apples </s>

<s> I like programming </s>

<s> Do I like using Python </s>

<s> I like using NLTK </s>

"""

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

trigrams = list(ngrams(tokens, 3))

context = ('do', 'i', 'like')

next\_words = [w3 for (w1, w2, w3) in trigrams if (w1, w2, w3)[:3] == context]

most\_common = Counter(next\_words).most\_common(1)

print("Most probable next word:", most\_common[0][0] if most\_common else "No suggestion")

5. import nltk

from nltk.stem import PorterStemmer, WordNetLemmatizer

nltk.download('wordnet')

nltk.download('omw-1.4')

words = ['Programming', 'Loving', 'Lovely', 'Kind']

stemmer = PorterStemmer()

lemmatizer = WordNetLemmatizer()

print(f"{'Word':<12}{'Stemmed':<12}{'Lemmatized':<12}")

print("-" \* 35)

for word in words: print(f"{word:<12}{stemmer.stem(word):<12}{lemmatizer.lemmatize(word.lower()):<12}")

6.

import nltk

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

sentences = [

"I need a flight from Atlanta.",

"Everything to permit us.",

"I would like to address the public on this issue.",

"We need your shipping address."

]

for sent in sentences:

tokens = nltk.word\_tokenize(sent)

tagged = nltk.pos\_tag(tokens)

print(f"\nSentence: {sent}")

print("POS Tags:", tagged)

7. from sumy.parsers.plaintext import PlaintextParser

from sumy.nlp.tokenizers import Tokenizer

from sumy.summarizers.lsa import LsaSummarizer

paragraph = """

Artificial Intelligence is transforming the world in unprecedented ways.

It enables machines to learn from data, adapt to new inputs, and perform tasks like humans.

AI is used in various sectors such as healthcare, finance, transportation, and education.

In healthcare, it helps in diagnosis, treatment personalization, and drug discovery.

In finance, AI algorithms detect fraud and automate trading.

Self-driving cars and traffic management are possible because of AI in transportation.

Educational tools powered by AI adapt to the learning speed of students.

AI also assists in language translation, image recognition, and personal assistants like Siri and Alexa.

However, it raises ethical concerns about privacy, bias, and job displacement.

Despite challenges, AI continues to be a revolutionary technology shaping the future.

"""

parser = PlaintextParser.from\_string(paragraph, Tokenizer("english"))

summarizer = LsaSummarizer()

summary = summarizer(parser.document, 3) # 3 lines summary

print("\nSummary:")

for sentence in summary:

print("-", sentence)

8.

import re

# Sample text with 3 emails and 2 phone numbers

text = """

Contact john.doe@example.com or jane\_smith123@gmail.com for inquiries.

Also reach out via help@service.co.in.

Call us at 9876543210 or 9123456789.

"""

# List of names to test pattern "S u \_ \_ \_"

names = ["Sunil", "Shyam", "Ankit", "Surjeet", "Sumit", "Subhi", "Surbhi", "Siddharth", "Sujan"]

print("=== Extract Email IDs ===")

emails = re.findall(r'\b[\w.-]+@[\w.-]+\.\w+\b', text)

print("Emails found:", emails)

print("\n=== Extract Phone Numbers ===")

phones = re.findall(r'\b\d{10}\b', text)

print("Phone numbers found:", phones)

print("\n=== Match Names with Pattern 'Su\_\_\_' ===")

name\_pattern = re.compile(r'^Su\w{3}$')

matched\_names = [name for name in names if name\_pattern.match(name)]

print("Matched names:", matched\_names)

print("\n=== re.search(), re.match(), re.sub(), re.compile(), re.findall() Demo ===")

sample = "The quick brown fox jumps over the lazy dog."

# re.search

search\_result = re.search(r'fox', sample)

print("re.search():", search\_result.group() if search\_result else "Not found")

# re.match

match\_result = re.match(r'The', sample)

print("re.match():", match\_result.group() if match\_result else "Not matched")

# re.sub

substituted\_text = re.sub(r'lazy', 'energetic', sample)

print("re.sub():", substituted\_text)

# re.compile and re.findall

compiled\_pattern = re.compile(r'\b\w{5}\b')

compiled\_matches = compiled\_pattern.findall(sample)

print("re.compile() & re.findall():", compiled\_matches)

all\_words = re.findall(r'\b\w+\b', sample)

print("re.findall() all words:", all\_words)

print("\n=== Match 'ab' followed by zero or more 'c' ===")

test1 = ["ab", "abc", "abcc", "ac", "abb"]

pattern1 = re.compile(r'abc\*')

matches1 = [t for t in test1 if pattern1.fullmatch(t)]

print("Matches:", matches1)

print("\n=== Match 'a' followed by zero or more 'bc' ===")

test2 = ["a", "abc", "abcbc", "abcbcbc", "ac"]

pattern2 = re.compile(r'a(bc)\*$')

matches2 = [t for t in test2 if pattern2.fullmatch(t)]

print("Matches:", matches2)

print("\n=== Match 'ab' followed by zero or one 'c' ===")

test3 = ["ab", "abc", "abcc", "ac", "abb"]

pattern3 = re.compile(r'ab(c)?$')

matches3 = [t for t in test3 if pattern3.fullmatch(t)]

print("Matches:", matches3)