EXP1: simple tokenizer

```
import re import nltk
nltk.download('punkt') nltk.download('punkt_tab')
from nltk.tokenize import word_tokenize
text = "Natural Language Processing (NLP) is fun!
 Let's tokenize this sentence." def basic_tokenizer(text):
    tokens = re.split(r'(\W+)', text) tokens =

    [t for t in tokens if t.strip()] return

    tokens

nltk_tokens = word_tokenize(text)

print("Input Text:")

print(text)

print("\nTokens using Basic Regex Tokenizer:")

print(basic_tokenizer(text))

print("\nTokens using NLTK Tokenizer:")

print(nltk_tokens)
```

EXP2:Spell correction

```
import numpy as np
import nltk
nltk.download('words')
from nltk.corpus import words
def edit_distance(str1, str2): m,
    n = len(str1), len(str2)
    dp = np.zeros((m+1, n+1), dtype=int) for i in
    range(m+1):
        dp[i][0] = i

    for j in range(n+1):
        dp[0][j] = j
    for i in range(1, m+1):
        for j in range(1,
        n+1):
            if str1[i-1] == str2[j-1]:
                dp[i][j] = dp[i-
                1][j-1]
            else:

                dp[i][j] = 1 + min(dp[i-1][j],

                                   dp[i][j-1],

                                   dp[i-1][j-1])

    return dp[m][n]
word_list = words.words()
misspelled_word = "speling"
distances = []
for w in word_list:

    d = edit_distance(misspelled_word, w)
    distances.append((d, w))
distances.sort()
top5 =
distances[:5]
print("Misspelled Word:", misspelled_word)
print("Top 5 Suggested Corrections:")
for dist, word in top5:

    print(f"{word} (Edit Distance: {dist})")
```

EXP3:parts of speech

```
import nltk
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger_eng')

from nltk import RegexpTagger, word_tokenize, pos_tag sentence = "The
quick brown fox jumps over the lazy dog" tokens =
word_tokenize(sentence)
patterns =
    [ (r'.*ing$',
    'VBG'),
    (r'.*ed$', 'VBD'),

    (r'.*es$', 'VBZ'),

    (r'^-?[0-9]+$', 'CD'),
    (r'.*', 'NN') ]
rule_based_tagger = RegexpTagger(patterns) rule_based_tags =
rule_based_tagger.tag(tokens) stochastic_tags = pos_tag(tokens)
print("Input Sentence:")
print(sentence)
print("\nRule-Based POS Tags:")
print(rule_based_tags)
print("\nStochastic POS Tags (using NLTK Perceptron Tagger):") print(stochastic_tags)

EXP 4:parsing sentence
import nltk
from nltk import CFG grammar = CFG.fromstring("""
S -> NP VP
NP -> Det N | Det Adj N | Det Adj Adj N VP -> V NP | V NP PP
PP -> P NP
Det -> 'the' | 'a'
Adj -> 'quick' | 'brown' | 'lazy' | 'small' N -> 'fox' | 'dog' | 'cat' | 'park'
V -> 'jumps' | 'runs' | 'sleeps' | 'sees' P -> 'in' | 'on' | 'over'
""")
parser = nltk.ChartParser(grammar)
sentence = ['the', 'quick', 'brown', 'fox', 'sees', 'a', 'dog'] for tree in parser.parse(sentence):
print("\nBracketed Tree format:\n") print(tree)
print("\nPretty Printed Tree:\n") tree.pretty_print()

EXP 5: DISAMBIGUATION
import nltk
from nltk.wsd import lesk
from nltk.corpus import wordnet as wn nltk.download('wordnet') nltk.download('omw-1.4')
sentence = "I went to the bank to deposit some money".split() target_word = "bank"
sense = lesk(sentence, target_word) print(f"\nSentence: {' '.join(sentence)}")
print(f"Target word: {target_word}\n") if

sense:

    print("Best Sense Found by Lesk Algorithm:") print("

                                                    ")
    _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
    print(f"Synset: {sense.name()}")

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

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

else:

    print("No sense could be found for the target word.")
```

EXP 6:WORD2VEC

```
import nltk import re

from gensim.models

import Word2Vec from nltk.tokenize

import word_tokenize
nltk.download("punkt")

text = """Natural Language Processing is a subfield of Artificial Intelligence.

Word embeddings like Word2Vec help computers understand semantic
meaning of words. Machine learning is widely used in NLP applications."""
def preprocess(text):
    text = text.lower()
    text = re.sub(r'[^a-z\s]', '', text)
    tokens = word_tokenize(text)
    return tokens
tokens = preprocess(text)
print("Tokens:", tokens)
sentences = [tokens]
model = Word2Vec(sentences, vector_size=50, window=3, min_count=1, sg=1, epochs=100)

print("\nSimilarity between 'language' and 'processing':", model.wv.similarity("language", "processing"))

print("Most similar to 'learning':", model.wv.most_similar("learning")) print("\nVector for 'nlp':\n",
model.wv['nlp'])
```