Experiment 3

☑ Experiment: Word Analysis and Word Generation using Python

🔍 Introduction:

This experiment demonstrates how to perform **word-level analysis** (like stemming, lemmatization, POS tagging, frequency distribution) and how to generate **random words** in Python.

🔠 Objective:

- Analyze a given input text for meaningful patterns and structure.
- Generate random words using basic string operations.

✨ Example Applications:

- **Word Analysis:** Useful in NLP tasks like sentiment analysis, summarization, chatbot processing, etc.
- **Word Generation:** Can be used for testing, generating passwords, CAPTCHAs, or game content.

🧠 Theory:

A) Word Analysis

This includes:

- **Tokenization** – Breaking text into words.
- **Stopword Removal** – Removing common words like "is", "the".
- **Stemming** – Reducing words to base form (e.g., "playing" → "play").
- **Lemmatization** – Reducing words to dictionary root (e.g., "better" → "good").
- **POS Tagging** – Identifying parts of speech (noun, verb, etc.).
- **Frequency Distribution** – Finding most repeated words.

Example:
Input: "This is Python Programming using NLTK."
Output: Tokens = ['This', 'is', 'Python', 'Programming', ...]

B) Word Generation

Generates random strings using random letters from the alphabet.

Example:
Output: fxgzeqt, uhprsg, mzlqw

⚙️ **Required Libraries:**

pip install nltk

🧪 **Program A: Word Analysis**

```
import nltk, random
from nltk import pos_tag, FreqDist, word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer

# Downloads
for pkg in ['punkt', 'stopwords', 'wordnet', 'averaged_perceptron_tagger']:
    nltk.download(pkg)

text = "This is Python Programming. We are using NLTK."

tokens = word_tokenize(text)
filtered = [w for w in tokens if w.lower() not in stopwords.words('english')]

print("Tokens:", tokens)
print("Filtered:", filtered)
```

```
print("Stemmed:", [PorterStemmer().stem(w) for w in filtered])
print("Lemmatized:", [WordNetLemmatizer().lemmatize(w) for w in filtered])
print("POS Tags:", pos_tag(filtered))
print("Most Common:", FreqDist(filtered).most_common(5))
```

Output:

```
Tokens: ['This', 'is', 'Python', 'Programming', '.', 'We', 'are', 'using', 'NLTK', '.']
Filtered: ['Python', 'Programming', '.', 'using', 'NLTK', '.']
Stemmed: ['python', 'program', '.', 'use', 'nltk', '.']
Lemmatized: ['Python', 'Programming', '.', 'using', 'NLTK', '.']
POS Tags: [('Python', 'NNP'), ('Programming', 'NNP'), ('.', '.'), ('using', 'VBG'), ('NLTK', 'NNP'), ('.', '.')]
Most Common: [('.', 2), ('Python', 1), ('Programming', 1), ('using', 1), ('NLTK', 1)]
```

🔁 **Program B: Word Generation**

```
import random
letters = "abcdefghijklmnopqrstuvwxyz"
for i in range(5):
    length = random.randint(5, 10)
    word = "".join(random.choice(letters) for i in range(length))
    print("Random Word:", word)
```

Output:

```
Random Word: xtvehzg
Random Word: cwofqyb
Random Word: bibicbb
Random Word: kibxqp
Random Word: btppdnvjl
```

📌 **Conclusion:**

- **Word Analysis** helps in understanding language patterns using NLP tools.
- **Word Generation** helps in creating random word data for testing or games.
- These techniques are important in fields like AI, ML, Text Mining, and Game Dev.

Experiment 7 b

🧪 Experiment: Generating N-Grams using Python and NLTK

✅ 🔍 **Introduction:**

In Natural Language Processing (NLP), understanding the context of words in a sentence is crucial. **N-Grams** help in analyzing the neighboring words by creating a sequence of **N continuous words**. This experiment demonstrates how to generate N-Grams (e.g., bigrams, trigrams) from input text using Python and the NLTK library.

🎯 **Objective:**

- To tokenize text into words.
- To generate and display N-Grams (bigrams, trigrams, etc.).
- To understand how N-Grams help capture word context in text.

📚 **Theory:**

An **N-Gram** is a contiguous sequence of **N items** from a given text. These items can be characters or words. N-Grams are widely used in:

- Text prediction,
- Spell checking,

- Machine translation,
- Language modeling.

Common Types:

- **Unigram (n=1)**: Single words → ['The', 'quick', 'brown']
- **Bigram (n=2)**: Pairs → [('The', 'quick'), ('quick', 'brown')]
- **Trigram (n=3)**: Triplets → [('The', 'quick', 'brown'), ('quick', 'brown', 'fox')]

✨ **Example:**

If input = "The quick brown fox"
Then:

- **Bigrams** = [('The', 'quick'), ('quick', 'brown'), ('brown', 'fox')]
- **Trigrams** = [('The', 'quick', 'brown'), ('quick', 'brown', 'fox')]

💻 **Python Program: N-Gram Generation**

```
import nltk
from nltk import ngrams, word_tokenize

nltk.download('punkt')
text = "The quick brown fox jumps over the lazy dog."
for g in ngrams(word_tokenize(text), 3):
    print(g)
```

✅ **Output:**

('The', 'quick', 'brown')

('quick', 'brown', 'fox')

('brown', 'fox', 'jumps')

('fox', 'jumps', 'over')

('jumps', 'over', 'the')

('over', 'the', 'lazy')

('the', 'lazy', 'dog')

('lazy', 'dog', '.')

📌 **Conclusion:**

- N-Grams are a fundamental concept in NLP used to model language patterns.
- This experiment helps visualize how sequences of words can be grouped and analyzed.
- Trigrams and other N-Grams are useful in applications like predictive text, summarization, and text classification.
