# Observation Notebook

**Date:** 2025-07-30

**Lab No:** 4

**Program Question:** *Design and implement a Python program to perform basic NLP tasks such as creating a positional index, generating a word matrix, preprocessing text, calculating edit distance, performing POS tagging, and word sense disambiguation.*

## Program Description

The program aims to provide an interactive platform to explore fundamental NLP tasks. Users can input documents and sentences to see how text is processed, indexed, and analyzed using Python. The tool offers visualizations and explanations for each step, helping users understand how NLP techniques are applied.

## Program Logic

* \*\*Positional Index:\*\*Tokenize documents and record the position of each word in every document using a dictionary structure.
* \*\*Word Matrix:\*\*Create a term-document incidence matrix indicating the presence or absence of each term in the documents.
* \*\*Preprocessing:\*\*Tokenize, stem, and lemmatize input text using NLTK. Count word frequencies and visualize them.
* \*\*Edit Distance:\*\*Calculate and visualize the Levenshtein distance between two words, showing the alignment and operations required for transformation.
* \*\*POS Tagging:\*\*Use NLTK’s POS tagging to train a simple HMM and display transition and emission probabilities.
* **Word Sense Disambiguation:** Apply the Lesk algorithm to determine the most appropriate sense for open-class words in a sentence.

**Libraries Used:**

* nltk for NLP tasks
* pandas and numpy for data manipulation
* streamlit for user interface
* collections for efficient data structures

**Data Types:**

* Lists, dictionaries, and data frames for managing and displaying results.

## Program

# Example: Code snippet for creating positional index  
from collections import defaultdict  
from nltk.tokenize import word\_tokenize  
  
def create\_positional\_index(docs):  
 index = defaultdict(lambda: defaultdict(list))  
 for doc\_id, text in docs.items():  
 tokens = word\_tokenize(text)  
 for pos, word in enumerate(tokens):  
 word = word.lower()  
 index[word][doc\_id].append(pos)  
 return index

*Further code for other modules is implemented similarly, following the logic described above.*

## Test Cases

| Module | Input Example | Expected Output | Actual Output |
| --- | --- | --- | --- |
| Positional Index | {“Doc1”: “I am a student.”, “Doc2”: “I was a student.”} | {‘i’: {‘Doc1’: [0], ‘Doc2’: [0]}, …} | {‘i’: {‘Doc1’: [0], ‘Doc2’: [0]}, …} |
| Word Matrix | [“student”, “I”, “was”] | Matrix with 1 for presence, 0 for absence | As expected |
| Edit Distance | “cat”, “cut” | 1 | 1 |
| POS Tagging | “The dog can chase the cat” | POS tags assigned | POS tags assigned |
| WSD | “The bank will not lend money to the poor risk.” | Disambiguated senses for ‘bank’, ‘risk’ | Disambiguated senses shown |

## Evaluation Comments