Absolutely! Let's go **very thoroughly**, explaining **every step, function, and output** in your code. I'll keep it structured so nothing is skipped.

1. Tokenization

Code:

```
from nltk.tokenize import word_tokenize

text = "Prof. Sasikala teaches Python."
print(word_tokenize(text))
```

Explanation:

- **Tokenization** is splitting text into smaller pieces called **tokens** (words, numbers, punctuation).
- `word_tokenize(text)` automatically splits words and punctuation.
- Input: `"Prof. Sasikala teaches Python."`
- Output: `['Prof.', 'Sasikala', 'teaches', 'Python', '.']`
  - `'Prof.'` → considered one token because of the period.
  - `'Python'` → recognized as a word.
  - `'.'` → punctuation is treated as a separate token.

✅ **Use:** Tokenization is the first step in NLP before stemming, lemmatization, or PoS tagging.

2. Stemming

Code:

```
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()

print(stemmer.stem("teaching"))
print(stemmer.stem("studies"))
```

Explanation:

- **Stemming** reduces a word to its **root form**, removing suffixes.
- `PorterStemmer()` is a commonly used algorithm.

- `"teaching"` → `"teach"` (suffix `"ing"` removed)
- `"studies"` → `"studi"` (suffix `"es"` removed, not a real word)

Output:

```
teach
studi
```

✅ **Use:** Useful for reducing different forms of a word to a single form for searching or analysis.

3. Lemmatization

Code:

```
from nltk.stem import WordNetLemmatizer
import nltk
nltk.download('wordnet')

lemmatizer = WordNetLemmatizer()
print(lemmatizer.lemmatize("studies", pos="v"))
```

Explanation:

- **Lemmatization** converts a word to its **dictionary/base form (lemma)** using vocabulary + POS tag.
- `pos="v"` tells lemmatizer that `"studies"` is a **verb**.
- `"studies"` → `"study"`
- Unlike stemming, lemmatization returns **real words**.

Output:

```
study
```

✅ **Use:** Better than stemming when you want meaningful words.

4. Part-of-Speech (PoS) Tagging

Code:

```
import nltk
nltk.download('averaged_perceptron_tagger')

tokens = word_tokenize("Engineer designs wearable devices.")
```

```
print(nltk.pos_tag(tokens))
```

Explanation:

- **PoS tagging** labels each word with its grammatical role.
- `nltk.pos_tag(tokens)` returns a list of tuples `(word, tag)`.

Output:

```
[('Engineer', 'NNP'), ('designs', 'VBZ'), ('wearable', 'JJ'),
('devices', 'NNS'), ('.', '.')]
```

Tag Meaning:

- `NNP` → Proper noun, singular (`Engineer`)
- `VBZ` → Verb, 3rd person singular present (`designs`)
- `JJ` → Adjective (`wearable`)
- `NNS` → Noun, plural (`devices`)
- `.` → punctuation

✅ **Use:** PoS tagging is necessary for lemmatization, NER, and syntactic parsing.

5. Named Entity Recognition (NER) with NLTK

Code:

```
nltk.download('maxent_ne_chunker')
nltk.download('words')

tagged = nltk.pos_tag(tokens)
entities = nltk.chunk.ne_chunk(tagged)
print(entities)
```

Explanation:

- **NER** identifies named entities (Person, Location, Organization).
- `ne_chunk(tagged)` creates a **tree structure** of entities.
- Here, the sentence `"Engineer designs wearable devices."` contains **no named entities**, so the output is just the POS-tagged tree.

Output (tree):

```
(S
  Engineer/NNP
  designs/VBZ
  wearable/JJ
  devices/NNS
  ./.)
```

✅ **Use:** Detects proper nouns and entity types for downstream tasks.

6. Accessing WordNet

Code:

```
from nltk.corpus import wordnet
syns = wordnet.synsets("teach")
print(syns[0].definition())
```

Explanation:

- WordNet allows you to access **synonyms, definitions, and relationships**.
- `synsets("teach")` → list of senses of the word `"teach"`.
- `syns[0].definition()` → meaning of the first sense.

Output:

```
impart skills or knowledge to
```

✅ **Use:** Useful for semantic analysis, NLP reasoning, and synonyms.

7. Named Entity Recognition with SpaCy

Code:

```
import spacy
nlp = spacy.load("en_core_web_sm")

text = """Prof. Sasikala visited Chennai, then flew to New
York for a conference.
She also stopped by Paris and Tokyo before returning to
India."""

doc = nlp(text)
places = [ent.text for ent in doc.ents if ent.label_ in
("GPE", "LOC")]
```

```
print("Identified Places:", set(places))
```

Explanation:

- SpaCy is more **modern and accurate** for NER.
- `nlp(text)` processes the text.
- `ent.label_` gives entity type:
  - `"GPE"` → Geo-Political Entity (country, city, state)
  - `"LOC"` → Location
- Extracted places: `{'Chennai', 'New York', 'Paris', 'Tokyo', 'India'}`

Output:

```
Identified Places: {'New York', 'Paris', 'Chennai', 'Tokyo',
'India'}
```

✅ **Use:** Automatically detects entities from text for information extraction.

8. Full NLTK Named Entity Extraction

Code:

```
import nltk
nltk.download('punkt')
nltk.download('maxent_ne_chunker')
nltk.download('words')
nltk.download('averaged_perceptron_tagger')

text = """Professor visited Chennai, then flew to New York
for a conference.
He also stopped by Paris and Tokyo before returning to
India."""

# 1. Sentence Tokenization
sentences = nltk.sent_tokenize(text)

# 2. Word Tokenization
tokenized_sentences = [nltk.word_tokenize(sent) for sent in
sentences]

# 3. PoS Tagging
```

```
tagged_sentences = [nltk.pos_tag(sent) for sent in
tokenized_sentences]

# 4. Named Entity Chunking
chunked_sentences = nltk.ne_chunk_sents(tagged_sentences,
binary=False)

# 5. Extract location entities
def extract_places(tree):
    places = []
    for subtree in tree:
        if hasattr(subtree, 'label') and subtree.label() in
('GPE', 'LOCATION'):
            place = " ".join([leaf[0] for leaf in
subtree.leaves()])
            places.append(place)
    return places

# 6. Collect all places
all_places = []
for tree in chunked_sentences:
    all_places.extend(extract_places(tree))

print("Identified Places:", set(all_places))
```

Step-by-Step Explanation:

1. **Sentence Tokenization:** `sent_tokenize` → splits text into sentences.
2. **Word Tokenization:** `word_tokenize` → splits sentences into words.
3. **PoS Tagging:** `pos_tag` → assigns grammatical roles to each word.
4. **Named Entity Chunking:** `ne_chunk_sents` → builds a **tree of entities**.
5. **Extract Places:** Loop through tree, select subtrees with label `GPE` or `LOCATION`, combine leaves into string.
6. **Collect Unique Places:** `set(all_places)` ensures unique locations.

Output:

```
Identified Places: {'New York', 'Paris', 'Chennai', 'Tokyo',
'India'}
```

✅ **Use:** Stepwise NER with NLTK for structured extraction of locations or other entity types.

Summary Table

Operati	Function	Output	Notes
Tokenization	`word_tokenize()`	`['Prof.', 'Sasikala',`	Splits text into words and
Stemming	`PorterStemmer().stem()`	`teach`, `studi`	Reduces words to root; may produce
Lemmatization	`WordNetLemmatizer().lemmatize("studies",`	`study`	Reduces to dictionary word
PoS	`pos_tag()`	`[('Engineer',`	Assigns part-of-
NER (NLTK)	`ne_chunk()`	`(S Engineer/ NNP ... )`	Detects named entities in tree
WordNe	`wordnet.synsets()`	`impart skills or`	Access meanings,
NER	`doc.ents`	`{'Chennai', 'New`	Modern, accurate
NER (NLTK	`ne_chunk_sents()` + extract	`{'Chennai', 'New York', ...}`	Stepwise extraction with

If you want, I can also **draw a visual flowchart showing the complete NLP pipeline from raw text → tokens → PoS → NER → extracted places**, which will make it very intuitive.

Do you want me to make that diagram?