Double-click (or enter) to edit

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

Python Program for Word and Sentence Tokenization using NLTK

```
Start coding or generate with AI.
```

```
pip install nltk
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.11/dist-packages (3.9.1)
Requirement already satisfied: click in /usr/local/lib/python3.11/dist-packages (from nltk) (8.2.1)
Requirement already satisfied: joblib in /usr/local/lib/python3.11/dist-packages (from nltk) (1.5.1)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.11/dist-packages (from nltk) (2024.11.6)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from nltk) (4.67.1)
```

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

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

```
import nltk
from nltk.tokenize import word_tokenize, sent_tokenize
```

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

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

```
text = """Natural Language Processing (NLP) is a sub-field of artificial intelligence.
It deals with the interaction between computers and humans using natural language."""
```

```
# Download required NLTK data (only needed once)
nltk.download('punkt_tab')

# Sentence Tokenization
sentence_tokens = sent_tokenize(text)
print("Sentence Tokenization:")
for i, sentence in enumerate(sentence_tokens, 1):
    print(f"{i}: {sentence}")

print("\n" + "-"*50 + "\n")
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
Sentence Tokenization:
1: Natural Language Processing (NLP) is a sub-field of artificial intelligence.
2: It deals with the interaction between computers and humans using natural language.

--------------------------------------------------
```

```
# Word Tokenization
word_tokens = word_tokenize(text)
print("Word Tokenization:")
print(word_tokens)
```

```
Word Tokenization:
['Natural', 'Language', 'Processing', '(', 'NLP', ')', 'is', 'a', 'sub-field', 'of', 'artificial', 'intelligence', '.', 'It', 'deals', 'with', 'the', 'int
```

2: Develop a Python script to annotate each word with its part of speech using NLTK and SpaCy. Compare the output and understand the role of syntactic context in language processing.

```
pip install nltk spacy
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.11/dist-packages (3.9.1)
Requirement already satisfied: spacy in /usr/local/lib/python3.11/dist-packages (3.8.7)
Requirement already satisfied: click in /usr/local/lib/python3.11/dist-packages (from nltk) (8.2.1)
Requirement already satisfied: joblib in /usr/local/lib/python3.11/dist-packages (from nltk) (1.5.1)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.11/dist-packages (from nltk) (2024.11.6)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from nltk) (4.67.1)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.11 in /usr/local/lib/python3.11/dist-packages (from spacy) (3.0.12)
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (1.0.5)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (1.0.13)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.0.11)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/lib/python3.11/dist-packages (from spacy) (3.0.10)
Requirement already satisfied: thinc<8.4.0,>=8.3.4 in /usr/local/lib/python3.11/dist-packages (from spacy) (8.3.6)
Requirement already satisfied: wasabi<1.2.0,>=0.9.1 in /usr/local/lib/python3.11/dist-packages (from spacy) (1.1.3)
Requirement already satisfied: srsly<3.0.0,>=2.4.3 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.5.1)
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.0.10)
Requirement already satisfied: weasel<0.5.0,>=0.1.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (0.4.1)
Requirement already satisfied: typer<1.0.0,>=0.3.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (0.16.0)
Requirement already satisfied: numpy>=1.19.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.0.2)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.32.3)
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.11.7)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from spacy) (3.1.6)
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from spacy) (75.2.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (25.0)
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (3.5.0)
Requirement already satisfied: language-data>=1.2 in /usr/local/lib/python3.11/dist-packages (from langcodes<4.0.0,>=3.2.0->spacy) (1.3.0)
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.11/dist-packages (from pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4->spacy) (0.7.0
Requirement already satisfied: pydantic-core==2.33.2 in /usr/local/lib/python3.11/dist-packages (from pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4->spacy) (2.33.2
Requirement already satisfied: typing-extensions>=4.12.2 in /usr/local/lib/python3.11/dist-packages (from pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4->spacy) (4.
Requirement already satisfied: typing-inspection>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4->spacy) (0.4
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.13.0->spacy) (3.4.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.13.0->spacy) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.13.0->spacy) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.13.0->spacy) (2025.7.14)
Requirement already satisfied: blis<1.4.0,>=1.3.0 in /usr/local/lib/python3.11/dist-packages (from thinc<8.4.0,>=8.3.4->spacy) (1.3.0)
Requirement already satisfied: confection<1.0.0,>=0.0.1 in /usr/local/lib/python3.11/dist-packages (from thinc<8.4.0,>=8.3.4->spacy) (0.1.5)
Requirement already satisfied: shellingham>=1.3.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0.0,>=0.3.0->spacy) (1.5.4)
Requirement already satisfied: rich>=10.11.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0.0,>=0.3.0->spacy) (13.9.4)
Requirement already satisfied: cloudpathlib<1.0.0,>=0.7.0 in /usr/local/lib/python3.11/dist-packages (from weasel<0.5.0,>=0.1.0->spacy) (0.21.1)
Requirement already satisfied: smart-open<8.0.0,>=5.2.1 in /usr/local/lib/python3.11/dist-packages (from weasel<0.5.0,>=0.1.0->spacy) (7.3.0.post1)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2->spacy) (3.0.2)
Requirement already satisfied: marisa-trie>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from language-data>=1.2->langcodes<4.0.0,>=3.2.0->spacy) (1.
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich>=10.11.0->typer<1.0.0,>=0.3.0->spacy) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich>=10.11.0->typer<1.0.0,>=0.3.0->spacy) (2.19.2
Requirement already satisfied: wrapt in /usr/local/lib/python3.11/dist-packages (from smart-open<8.0.0,>=5.2.1->weasel<0.5.0,>=0.1.0->spacy) (1.17.2)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich>=10.11.0->typer<1.0.0,>=0.3.0->spac
```

```
python -m nltk.downloader punkt averaged_perceptron_tagger
```

```
  File "/tmp/ipython-input-6-101667130.py", line 1
    python -m nltk.downloader punkt averaged_perceptron_tagger
              ^
SyntaxError: invalid syntax
```

```
Start coding or generate with AI.
```

The cell `4M7DNxv-qBnP` produced a `SyntaxError` because it contains a shell command that was not prefixed with `!`. The `!` prefix is required to execute shell commands directly within a code cell in Google Colab.

```
!python -m nltk.downloader punkt averaged_perceptron_tagger
```

```
<frozen runpy>:128: RuntimeWarning: 'nltk.downloader' found in sys.modules after import of package 'nltk', but prior to execution of 'nltk.downloader'; th
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

```
python -m spacy download en_core_web_sm
```

```
  File "/tmp/ipython-input-8-581980377.py", line 1
    python -m spacy download en_core_web_sm
              ^
SyntaxError: invalid syntax
```

```
Start coding or generate with AI.
```

NLP 3 Program ; Use NLTK to clean and normalize raw text by splitting it into tokens. Apply stemming to reduce words to root form and lemmatization for grammatically correct base forms. This is a key step before any NLP task.

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords, wordnet
from nltk.stem import PorterStemmer, WordNetLemmatizer
```

```
import string

# Download required NLTK data (only needed once)
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
nltk.download('punkt_tab')
nltk.download('averaged_perceptron_tagger_eng')

# Sample raw text
raw_text = """
Natural Language Processing (NLP) is a crucial part of AI. It involves cleaning, tokenizing, stemming,
and lemmatizing raw texts before applying machine learning algorithms.
"""

# 1. Lowercase the text
text = raw_text.lower()

# 2. Remove punctuation
text = text.translate(str.maketrans('', '', string.punctuation))

# 3. Tokenize the text
tokens = word_tokenize(text)

# 4. Remove stopwords
stop_words = set(stopwords.words('english'))
filtered_tokens = [word for word in tokens if word not in stop_words]

# 5. Initialize stemmer and lemmatizer
stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()

# 6. Stemming
stemmed_tokens = [stemmer.stem(word) for word in filtered_tokens]
```

```python
# 7. Function to get WordNet POS tag for lemmatization
def get_wordnet_pos(word):
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {'J': wordnet.ADJ,
                'N': wordnet.NOUN,
                'V': wordnet.VERB,
                'R': wordnet.ADV}
    return tag_dict.get(tag, wordnet.NOUN)

# 8. Lemmatization with POS tagging
lemmatized_tokens = [lemmatizer.lemmatize(word, get_wordnet_pos(word)) for word in filtered_tokens]

# 9. Print results
print("Original Tokens:", tokens)
print("Filtered Tokens:", filtered_tokens)
print("Stemmed Tokens:", stemmed_tokens)
print("Lemmatized Tokens:", lemmatized_tokens)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger_eng.zip.
Original Tokens: ['natural', 'language', 'processing', 'nlp', 'is', 'a', 'crucial', 'part', 'of', 'ai', 'it', 'involves', 'cleaning', 'tokenizing', 'stemm
Filtered Tokens: ['natural', 'language', 'processing', 'nlp', 'crucial', 'part', 'ai', 'involves', 'cleaning', 'tokenizing', 'stemming', 'lemmatizing', 'r
Stemmed Tokens: ['natur', 'languag', 'process', 'nlp', 'crucial', 'part', 'ai', 'involv', 'clean', 'token', 'stem', 'lemmat', 'raw', 'text', 'appli', 'mac
Lemmatized Tokens: ['natural', 'language', 'processing', 'nlp', 'crucial', 'part', 'ai', 'involves', 'cleaning', 'tokenizing', 'stem', 'lemmatizing', 'raw
```

```python
import nltk
import spacy

# Download required NLTK resources
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger_eng') # Download the correct resource
nltk.download('punkt_tab')

# Load SpaCy English model
nlp = spacy.load("en_core_web_sm")

# Input sentence
sentence = "The quick brown fox jumps over the lazy dog near the river bank."

# ---- NLTK POS tagging ----
nltk_tokens = nltk.word_tokenize(sentence)
nltk_pos_tags = nltk.pos_tag(nltk_tokens)

# ---- SpaCy POS tagging ----
doc = nlp(sentence)
spacy_pos_tags = [(token.text, token.pos_, token.tag_) for token in doc]

# ---- Display comparison ----
print(f"{'Word':<15}{'NLTK POS':<15}{'spaCy POS':<15}{'spaCy Fine POS':<20}")
print("-" * 65)
for ((nltk_word, nltk_tag), (spacy_word, spacy_pos, spacy_tag)) in zip(nltk_pos_tags, spacy_pos_tags):
    print(f"{nltk_word:<15}{nltk_tag:<15}{spacy_pos:<15}{spacy_tag:<20}")
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger_eng.zip.
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
Word           NLTK POS       spaCy POS      spaCy Fine POS
-----------------------------------------------------------------
The            DT             DET            DT
```

```
quick          JJ             ADJ            JJ
brown          NN             ADJ            JJ
fox            NN             NOUN           NN
jumps          VBZ            VERB           VBZ
over           IN             ADP            IN
the            DT             DET            DT
lazy           JJ             ADJ            JJ
dog            NN             NOUN           NN
near           IN             ADP            IN
the            DT             DET            DT
river          NN             NOUN           NN
bank           NN             NOUN           NN
.              .              PUNCT          .
```

```
Start coding or generate with AI.
```

22nd Aug 2025 unit 5 NLTK

NLTK means Natural Language Toolkit.

It is a Python library used for working with human language data (text). NLTK provides easy-to-use tools for:

Text preprocessing – tokenization, stemming, lemmatization, stopword removal

Part-of-speech tagging – identifying nouns, verbs, adjectives, etc.

Named Entity Recognition (NER) – finding names, places, organizations in text

Chunking and parsing – analyzing grammatical structure

Corpora and lexical resources – access to WordNet and many text datasets

```
# Install NLTK (if not already installed)
!pip install nltk
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.12/dist-packages (3.9.1)
Requirement already satisfied: click in /usr/local/lib/python3.12/dist-packages (from nltk) (8.2.1)
Requirement already satisfied: joblib in /usr/local/lib/python3.12/dist-packages (from nltk) (1.5.1)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.12/dist-packages (from nltk) (2024.11.6)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (from nltk) (4.67.1)
```

```
# Import required libraries
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk.tokenize import word_tokenize
from nltk import pos_tag
```

When you use **NLTK**, some resources (like tokenizers, stopwords, taggers, corpora, or dictionaries) are **not installed automatically**. So, we need to **download them once** using `nltk.download()`.

---

1. `nltk.download('punkt')`

- **What it is**: Pre-trained tokenizer models.
- **Use**: Splits text into sentences and words (tokenization).
- Example: `"I love NLP." → ["I", "love", "NLP", "."]`

---

2. `nltk.download('stopwords')`

- **What it is**: A list of common words (stopwords) in many languages like English, French, German, etc.
- **Use**: Helps remove frequent but less meaningful words like *"is"*, *"the"*, *"an"*, *"and"*.

---

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

- **What it is**: A **part-of-speech (POS) tagger** trained on English.
- **Use**: Identifies the role of each word in a sentence (noun, verb, adjective, etc.).
- Example: `"Dogs bark" → [("Dogs", NNS), ("bark", VBP)]`

---

4. `nltk.download('wordnet')`

- **What it is**: A large English **lexical database**.
- **Use**: Supports synonyms, antonyms, definitions, and lemmatization (reducing words to their base form).
- Example: `"running" → "run"`

---

5. `nltk.download('omw-1.4')`

- **What it is**: Open Multilingual WordNet.
- **Use**: Provides translations and multilingual support for WordNet.

---

6. `nltk.download('punkt_tab')`

- **What it is**: Additional data for **sentence tokenization** (tables for abbreviations, punctuation rules, etc.).
- **Use**: Makes tokenization more accurate when splitting text into sentences.

✅ **In short:**

- `punkt` → tokenizer
- `stopwords` → list of common words to remove
- `averaged_perceptron_tagger` → part-of-speech tagging
- `wordnet` → dictionary/lemmatizer
- `omw-1.4` → multilingual WordNet
- `punkt_tab` → tokenizer helper tables

```
# Download necessary NLTK data files
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('punkt_tab') # Download the missing resource
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
True
```

```
# Input paragraph
text = """Natural Language Processing (NLP) is a field of Artificial Intelligence
that focuses on the interaction between computers and humans using natural language.
It involves various techniques such as tokenization, stemming, lemmatization, and part-of-speech tagging.""
```

word_tokenize

This function from NLTK splits text into tokens (small units).

Tokens can be words, numbers, or punctuation marks.

tokens = word_tokenize(text)

Here, text is your input string (a sentence, paragraph, or document).

The function processes it and returns a Python list of tokens.

```
# Tokenize text
tokens = word_tokenize(text)
```

```
# Tokenize text
tokens = word_tokenize(text)
```

stop_words = set(stopwords.words('english'))

NLTK has a stopwords list for many languages.

Here, we load the English stopwords.

Example of stopwords: "is", "the", "and", "in", "on", "at".

Converting it into a set makes lookups faster than a list.

2. filtered_tokens = [word for word in tokens if word.lower() not in stop_words]

This is a list comprehension that loops through all tokens.

It keeps only those tokens that are not in the stopwords list.

.lower() ensures case-insensitivity (so "The" and "the" are treated the same).

```
# Remove stopwords
stop_words = set(stopwords.words('english'))
filtered_tokens = [word for word in tokens if word.lower() not in stop_words]
```

```
# Remove stopwords
stop_words = set(stopwords.words('english'))
filtered_tokens = [word for word in tokens if word.lower() not in stop_words]
```

1. stemmer = PorterStemmer()

PorterStemmer is a popular stemming algorithm in NLTK.

Stemming means reducing words to their root/base form by chopping off suffixes/prefixes.

It is rule-based (not always producing real words).

2. stemmed_words = [stemmer.stem(word) for word in filtered_tokens]

This line loops over all words in filtered_tokens.

For each word, it applies stemmer.stem(word) to get the stemmed/root form.

The results are collected in a new list stemmed_words.

```
# Perform stemming
stemmer = PorterStemmer()
stemmed_words = [stemmer.stem(word) for word in filtered_tokens]
```

```
# Perform stemming
stemmer = PorterStemmer()
stemmed_words = [stemmer.stem(word) for word in filtered_tokens]
```

Step-by-step Explanation

1. lemmatizer = WordNetLemmatizer()

Creates a lemmatizer object using WordNet (the lexical database you downloaded earlier).

Lemmatization reduces words to their dictionary form (lemma).

Unlike stemming, it always produces a valid word.

2. lemmatized_words = [lemmatizer.lemmatize(word) for word in filtered_tokens]

This loops over each word in filtered_tokens.

For each word, it applies lemmatizer.lemmatize(word) to get its base dictionary form.

The result is stored in a new list lemmatized_words.

```
# Perform lemmatization
lemmatizer = WordNetLemmatizer()
lemmatized_words = [lemmatizer.lemmatize(word) for word in filtered_tokens]
```

Step-by-step Explanation

1. nltk.download('averaged_perceptron_tagger_eng')

Downloads the Averaged Perceptron Tagger model (English version).

This is a pre-trained statistical model that assigns parts of speech (POS) to words.

POS = grammatical role → noun, verb, adjective, adverb, etc.

2. pos_tags = pos_tag(filtered_tokens)

pos_tag() is an NLTK function that takes a list of words (tokens) and assigns POS tags.

Each word is returned as a tuple: (word, POS_tag).

```
# Download necessary NLTK data files for POS tagging (if not already downloaded)
import nltk
nltk.download('averaged_perceptron_tagger_eng')

# Part-of-Speech Tagging
pos_tags = pos_tag(filtered_tokens)
```

```
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger_eng.zip.
```

```
# Display Results
print("\nOriginal Tokens:\n", tokens)
print("\nAfter Stopword Removal:\n", filtered_tokens)
print("\nAfter Stemming:\n", stemmed_words)
print("\nAfter Lemmatization:\n", lemmatized_words)
print("\nPOS Tags:\n", pos_tags)
```

```
Original Tokens:
 ['Natural', 'Language', 'Processing', '(', 'NLP', ')', 'is', 'a', 'field', 'of', 'Artificial', 'Intelligence', 'that', 'focuses', 'on', 'the', 'interacti

After Stopword Removal:
 ['Natural', 'Language', 'Processing', '(', 'NLP', ')', 'field', 'Artificial', 'Intelligence', 'focuses', 'interaction', 'computers', 'humans', 'using', '

After Stemming:
 ['natur', 'languag', 'process', '(', 'nlp', ')', 'field', 'artifici', 'intellig', 'focus', 'interact', 'comput', 'human', 'use', 'natur', 'languag', '.',

After Lemmatization:
 ['Natural', 'Language', 'Processing', '(', 'NLP', ')', 'field', 'Artificial', 'Intelligence', 'focus', 'interaction', 'computer', 'human', 'using', 'natu

POS Tags:
 [('Natural', 'JJ'), ('Language', 'NNP'), ('Processing', 'NNP'), ('(', '('), ('NLP', 'NNP'), (')', ')'), ('field', 'NN'), ('Artificial', 'NNP'), ('Intelli
```

```
Start coding or generate with AI.
```

NER (Named Entity Recognition): Identifies proper nouns like Apple, U.K., John Smith, Google.

Chunking: Groups words into meaningful phrases (e.g., Noun Phrases).

```
# Import libraries
import nltk
from nltk import pos_tag, ne_chunk
from nltk.tokenize import word_tokenize
from nltk.chunk import RegexpParser
```

```
# Download necessary data
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
```

```python
nltk.download('maxent_ne_chunker')
nltk.download('words')
nltk.download('maxent_ne_chunker_tab') # Download the missing resource for NER
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package maxent_ne_chunker is already up-to-date!
[nltk_data] Downloading package words to /root/nltk_data...
[nltk_data]   Package words is already up-to-date!
[nltk_data] Downloading package maxent_ne_chunker_tab to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping chunkers/maxent_ne_chunker_tab.zip.
True
```

```python
# Input text
text = """Apple is looking at buying U.K. startup for $1 billion.
John Smith, a software engineer, lives in New York and works at Google."""
```

```python
# Tokenize
tokens = word_tokenize(text)
```

```python
# POS Tagging
pos_tags = pos_tag(tokens)
```

```python
# Named Entity Recognition (NER)
ner_tree = ne_chunk(pos_tags)
print("Named Entity Recognition (NER):")
print(ner_tree)
```

```
Named Entity Recognition (NER):
(S
  (GPE Apple/NNP)
  is/VBZ
  looking/VBG
  at/IN
  buying/VBG
  U.K./NNP
  startup/NN
  for/IN
  $/$
  1/CD
  billion/CD
  ./.
  (PERSON John/NNP Smith/NNP)
  ,/,
  a/DT
  software/NN
  engineer/NN
  ,/,
  lives/VBZ
  in/IN
  (GPE New/NNP York/NNP)
  and/CC
  works/VBZ
  at/IN
  (ORGANIZATION Google/NNP)
  ./.)
```

```python
# Chunking: Define Grammar for Noun Phrases (NP)
grammar = "NP: {<DT>?<JJ>*<NN>}"  # Determiner + adjectives (optional) + Noun
chunk_parser = RegexpParser(grammar)
chunk_tree = chunk_parser.parse(pos_tags)
```

```python
print("\nChunked Phrases:")
print(chunk_tree)
```

```

Chunked Phrases:
(S
  Apple/NNP
  is/VBZ
  looking/VBG
  at/IN
  buying/VBG
  U.K./NNP
  (NP startup/NN)
  for/IN
  $/$
```

```
        1/CD
        billion/CD
        ./.
        John/NNP
        Smith/NNP
        ,/,
        (NP a/DT software/NN)
        (NP engineer/NN)
        ,/,
        lives/VBZ
        in/IN
        New/NNP
        York/NNP
        and/CC
        works/VBZ
        at/IN
        Google/NNP
        ./.)
```