





NLTK (Natural Language Toolkit)

```
pip install nltk
```

- It's a **Python library** made to help people **learn, explore, and use NLP tools**.
- NLTK is great for **beginners** and for **academic learning**, especially for understanding **basic NLP concepts** like:
 - Tokenization
 - Stemming
 - Lemmatization
 - POS tagging
 - Named Entity Recognition (NER)
 - Parsing
 - Stop word removal

Why Use NLTK?

Feature	What It Offers
 Educational	Easy to understand; used in NLP courses and books
 Rich Resources	Comes with text corpora , dictionaries, grammars
 Experimentation	Great for trying out basic NLP concepts
 Tools	Has tools for tokenizing, tagging, parsing, etc.

Download NLTK's internal resources:

```
import nltk
nltk.download('all') # ⚠ Downloads ~3GB of data
```



OR download selected datasets:

```
nltk.download('punkt')    # For tokenization
nltk.download('stopwords') # Common English words to filter
nltk.download('wordnet')   # For lemmatization
nltk.download('averaged_perceptron_tagger') # For POS tagging
```

Key Features of NLTK

1 Tokenization (Breaking text into words or sentences)

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

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

```
text = "Hello Sir! Welcome to NLP. Let's start learning."
```

```
print(sent_tokenize(text)) # Splits into sentences
print(word_tokenize(text)) # Splits into words
```

```
['Hello Sir!', 'Welcome to NLP.', "Let's start learning."]
['Hello', 'Sir', '!', 'Welcome', 'to', 'NLP', '.', 'Let', "'s", 'start', 'learning', '.']
```

 **Tokenization is the first step in almost any NLP pipeline.**

2 Stop Words Removal

Stop words = very common words (e.g. **"is"**, **"the"**, **"and"**) which often add little meaning.

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

```
from nltk.corpus import stopwords  
from nltk.tokenize import word_tokenize
```

```
text = "This is a simple example to show stopword removal."  
words = word_tokenize(text)
```

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

```
filtered = [word for word in words if word.lower() not in stop_words]  
print(filtered)
```

```
['simple', 'example', 'show', 'stopword', 'removal', '.']
```

To remove the punctuation, use `and word.isalpha()` 📌

```
from nltk.corpus import stopwords
```

```
words= word_tokenize(paragraph)
```

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

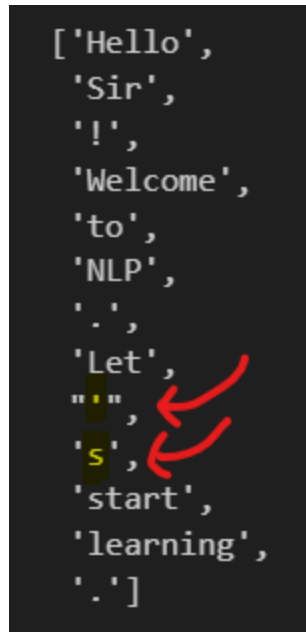
```
[word for word in words if word.lower() not in stop_words and word.isalpha()]
```

```
stopwords.words('english')  
✓ 0.0s  
['a',  
 'about',  
 'above',  
 'after',  
 'again',  
 'against',  
 'ain',  
 'all',  
 'am',  
 'an',  
 'and',  
 'any',  
 'are',  
 'aren',  
 "aren't",  
 'as',  
 'at',  
 'be',  
 'because',  
 'been',  
 'before',
```

```
from nltk import wordpunct_tokenize
```

```
wordpunct_tokenize(text)
```

```
['Hello',  
 'Sir',  
 '!',  
 'Welcome',  
 'to',  
 'NLP',  
 '.',  
 'Let',  
 '"',  
 's',  
 'start',  
 'learning',  
 '.']
```



- Here, 's' is also splitted

3

Stemming (Reducing words to root)

- **Eliminates prefixes and suffixes from words**
 - "chocolates" → "chocolate"
 - "retrieval" → "retrieve."
- Cuts off prefixes/suffixes to get root form (may not be a valid word).
 - **Example: "running" → "run", "happiness" → "happi"**
- **Speed:** Faster (rule-based).
- **Accuracy:** Less accurate (can produce non-words).
- **Use Case:** Information retrieval, search engines.

```
from nltk.stem import PorterStemmer
```

```
stemmer = PorterStemmer()
words = ["running", "runs", "happiness", "fairly"]

for word in words:
    print(f"{word} → {stemmer.stem(word)}")
```

```
running → run
runs → run
happiness → happi
fairly → fairli
```

 Stemming is **rule-based** and sometimes **rough or imprecise**.

```
from nltk.stem import PorterStemmer, WordNetLemmatizer

stemmer = PorterStemmer()
print(stemmer.stem("happiness")) # "happi"

lemmatizer = WordNetLemmatizer()
print(lemmatizer.lemmatize("happiness", pos='n')) # "happiness"
```

```
happi
happiness
```

Stemming + Stop word removal:

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import SnowballStemmer

text = """Thank you all so very much. Thank you to the Academy. Thank you t
o all of you in this room. I have to congratulate the other incredible nominees t
```

his year. The Revenant was the product of the tireless efforts of an unbelievable cast and crew. Making The Revenant was about man's relationship to the natural world."""

```
words = word_tokenize(text)
```

```
stemmer = SnowballStemmer('english')
```

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

```
filtered = [stemmer.stem(word.lower()) for word in words if word.lower() not in stop_words and word.isalpha()]  
print(filtered)
```

```
['thank', 'much', 'thank', 'academi', 'thank', 'room', 'congratul', 'incred',  
'nomine', 'year', 'reven', 'product', 'tireless', 'effort', 'unbeliev', 'cast', 'crew',  
'make', 'reven', 'man', 'relationship', 'natur', 'world']
```

4 Lemmatization (More accurate than stemming)

- Lemmatization is the process of reducing a word to its base form, but unlike stemming, **it takes into account the context of the word**, and it produces a valid word, unlike stemming which may produce a non-word as the root form.
- The output is called → **"Lemma"**
- Uses vocabulary & grammar rules to return base/dictionary form (lemma).
 - **Example:** "running" → "run", "better" → "good"
- **Speed:** Slower (requires POS tags & dictionary lookup).
- **Accuracy:** More accurate (produces valid words).
- **Use Case:** Chatbots, text analysis, where meaning matters.

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

```
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()
words = ["running", "flies", "better", "feet","happiness"]

for word in words:
    print(f"{word} → {lemmatizer.lemmatize(word)}")
```

```
running → running
flies → fly
better → better
feet → foot
happiness → happiness
```



Note: For accurate results, specify POS tag:

pos : str

- The Part Of Speech tag. Valid options are
 - **"n"** for nouns
 - **"v"** for verbs
 - **"a"** for adjectives
 - **"r"** for adverbs
 - **"s"** for satellite adjectives

```
lemmatizer.lemmatize('running', pos='v') # 'run'
```

Output: 'run'



One major difference with stemming is that lemmatize takes a part of speech parameter, "pos".

If not supplied, the default is "noun."

Previous example with lemmatize instead of stemming:

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import SnowballStemmer
```

```
lemmatizer = WordNetLemmatizer()
```

```
text = """Thank you all so very much. Thank you to the Academy. Thank you to  
all of you in this room. I have to congratulate the other incredible nominees this  
year. The Revenant was the product of the tireless efforts of an unbelievable  
cast and crew. Making The Revenant was about man's relationship to the natural  
world."""
```

```
words = word_tokenize(text)
```

```
stemmer = SnowballStemmer('english')
```

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

```
filtered = [lemmatizer.lemmatize(word.lower()) for word in words if word.lower()  
() not in stop_words and word.isalpha()]  
print(filtered)
```

```
['thank', 'much', 'thank', 'academy', 'thank', 'room', 'congratulate', 'incredible',  
'nominee', 'year', 'revenant', 'product', 'tireless', 'effort', 'unbelievable', 'cast',  
'crew', 'making', 'revenant', 'man', 'relationship', 'natural', 'world']
```

Result with stemmer:

['thank', 'much', 'thank', 'academi', 'thank', 'room', 'congratul', 'incred', 'nomine', 'year', 'reven', 'product', 'tireless', 'effort', 'unbeliev', 'cast', 'crew', 'make', 'reven', 'man', 'relationship', 'natur', 'world']

5 Part of Speech (POS) Tagging

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

```
from nltk import pos_tag
from nltk.tokenize import word_tokenize

text = "The quick brown fox jumps over the lazy dog."
tokens = word_tokenize(text)
tags = pos_tag(tokens)

print(tags)
```

```
[('The', 'DT'), ('quick', 'JJ'), ('brown', 'NN'), ('fox', 'NN'), ('jumps', 'VBZ'), ('over', 'IN'),
```

Sample tags:

- **NN** = Noun
- **VB** = Verb
- **JJ** = Adjective
- **RB** = Adverb

```
pos_tag(['the'])
```

Output:
[('the', 'DT')]

 Full tag list: [`nltk.help.upenn_tagset()`]

```
nltk.download('tagsets')  
nltk.help.upenn_tagset()
```

1. CC Coordinating conjunction
2. CD Cardinal number
3. DT Determiner
4. EX Existentialthere
5. FW Foreign word
6. IN Preposition or subordinating conjunction
7. JJ Adjective
8. JJR Adjective, comparative
9. JJS Adjective, superlative
10. LS List item marker
11. MD Modal
12. NN Noun, singular or mass
13. NNS Noun, plural
14. NNP Proper noun, singular
15. NNPS Proper noun, plural
16. PDT Predeterminer
17. POS Possessive ending
18. PRP Personal pronoun
19. PRP\$ Possessive pronoun
20. RB Adverb
21. RBR Adverb, comparative
22. RBS Adverb, superlative
23. RP Particle
24. SYM Symbol
25. TOTO
26. UH Interjection

- 27. VB Verb, base form
- 28. VBD Verb, past tense
- 29. VBG Verb, gerund or present participle
- 30. VBN Verb, past participle
- 31. VBP Verb, non-3rd person singular present
- 32. VBZ Verb, 3rd person singular present
- 33. WDT Wh-determiner
- 34. WP Wh-pronoun
- 35. WP\$ Possessive wh-pronoun
- 36. WRB Wh-adverb

Named Entity Recognition (NER)



Recognizes names of people, places, organizations, etc.

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

```
from nltk import ne_chunk, pos_tag
from nltk.tokenize import word_tokenize

sentence = "Barack Obama was born in Hawaii."
tokens = word_tokenize(sentence)
tags = pos_tag(tokens)

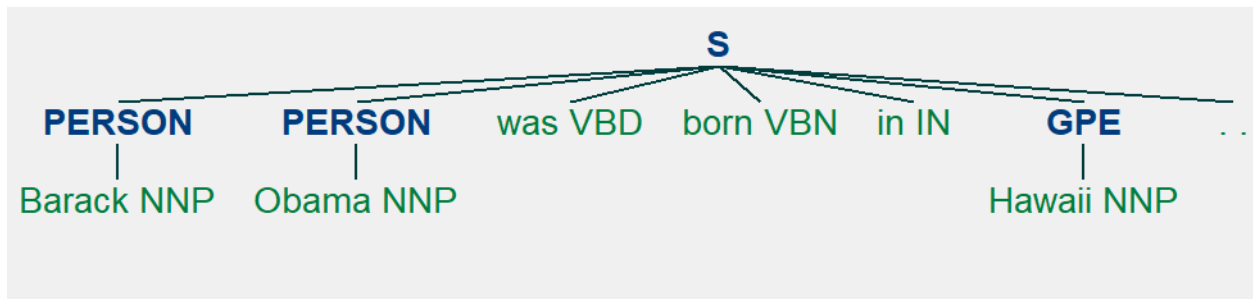
tree = ne_chunk(tags)
print(tree)
```

```

(S
  (PERSON Barack/NNP)
  (PERSON Obama/NNP)
  was/VBD
  born/VBN
  in/IN
  (GPE Hawaii/NNP)
  ./.)

```

tree.draw()



Common NER Tags (Standard Categories)

Tag	Description	Examples
PERSON	People, including fictional	"Marie Curie", "Sherlock Holmes"
ORGANIZATION	Companies, institutions, teams	"NASA", "Harvard University"
LOCATION	Physical locations (non-GPE)	"Mount Everest", "Pacific Ocean"
GPE	Geo-political entities (countries, cities)	"Japan", "Berlin"
FACILITY	Man-made structures (buildings, airports)	"Eiffel Tower", "JFK Airport"
DATE	Absolute or relative dates	"January 2025", "next week"
TIME	Times of day/duration	"3:30 PM", "two hours"
MONEY	Monetary values	"\$5 million", "€100"

Tag	Description	Examples
PERCENT	Percentages	"75%", "ten percent"

Key Notes

1. GPE vs. LOCATION:

- **GPE** = Political entities (cities/countries)
- **LOCATION** = Natural/physical places (rivers, mountains)

2. No **NORP** / **PRODUCT** : Unlike spaCy, NLTK doesn't distinguish nationalities or products.

3. Case Sensitivity:

NLTK's tagger is case-sensitive. Proper nouns (capitalized) are more likely to be tagged as entities.

4. Dependency:

Requires `pos_tag()` first (as shown in the example).

Text Corpus & Examples

```
nltk.download('gutenberg')
```

NLTK includes many linguistic corpora:

```
from nltk.corpus import gutenberg
gutenberg.fileids() # List of included texts
```

```
['austen-emma.txt',  
 'austen-persuasion.txt',  
 'austen-sense.txt',  
 'bible-kjv.txt',  
 'blake-poems.txt',  
 'bryant-stories.txt',  
 'burgess-busterbrown.txt',  
 'carroll-alice.txt',  
 'chesterton-ball.txt',  
 'chesterton-brown.txt',  
 'chesterton-thursday.txt',  
 'edgeworth-parents.txt',  
 'melville-moby_dick.txt',  
 'milton-paradise.txt',  
 'shakespeare-caesar.txt',  
 'shakespeare-hamlet.txt',  
 'shakespeare-macbeth.txt',  
 'whitman-leaves.txt']
```

NLTK includes sample texts you can use to practice.

```
from nltk.corpus import gutenberg  
print(gutenberg.fileids()) # List of available books  
  
sample = gutenberg.raw('austen-emma.txt')  
print(sample[:500])
```

```
['austen-emma.txt', 'austen-persuasion.txt', 'austen-sense.txt', 'bible-kjv.txt', 'blake-poems.txt',  
[Emma by Jane Austen 1816]
```

VOLUME I

CHAPTER I

Emma Woodhouse, handsome, clever, and rich, with a comfortable home and happy disposition, seemed to unite some of the best blessings of existence; and had lived nearly twenty-one years in the world with very little to distress or vex her.

She was the youngest of the two daughters of a most affectionate, indulgent father; and had, in consequence of her sister's marriage, been mistress of his house from a very early period. Her mother had died t

Summary Table of Key Functions

Task	NLTK Function / Module
Tokenization	<code>word_tokenize</code> , <code>sent_tokenize</code>
Stopwords	<code>nltk.corpus.stopwords</code>
Stemming	<code>PorterStemmer</code> , <code>LancasterStemmer</code>
Lemmatization	<code>WordNetLemmatizer</code>
POS Tagging	<code>pos_tag</code>
NER	<code>ne_chunk</code>
Corpora Access	<code>nltk.corpus</code> (e.g. <code>gutenberg</code> , <code>reuters</code>)

Advantages of NLTK

- Excellent for education and research
- Wide range of linguistic tools
- Good documentation and community support
- Many built-in corpora for experimentation

Limitations

- Not as fast as some modern alternatives (like spaCy) for production

- Somewhat dated algorithms compared to deep learning approaches
- Requires downloading additional data packages

TreebankWordTokenizer

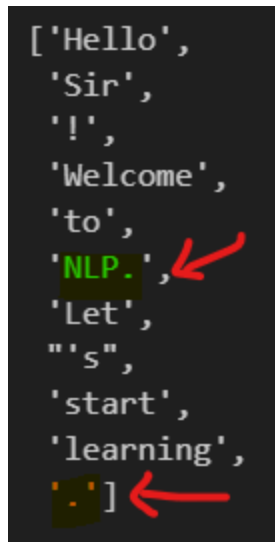
- **Full stop** (*except the last full stop*) **will not be considered as a separate word.**

```
from nltk.tokenize import TreebankWordTokenizer

tokenizer = TreebankWordTokenizer()

text = "Hello Sir! Welcome to NLP. Let's start learning."

tokenizer.tokenize(text)
```



```
['Hello',  
'Sir',  
'!',  
'Welcome',  
'to',  
'NLP.',  
'Let',  
's',  
'start',  
'learning',  
'']
```

Snowball Stemmer (Porter2)

- Improved version of the classic **Porter Stemmer**, supporting **multiple languages** and offering better stemming accuracy.
- It's widely used in **information retrieval** and **text preprocessing**.

Key Features

- ✓ **Supports multiple languages** (English, French, Spanish, German, etc.)
- ✓ **More aggressive than Porter Stemmer** (better for search engines)
- ✓ **Lightweight and fast**

```
from nltk.stem import SnowballStemmer

# Initialize for English
stemmer = SnowballStemmer("english")

words = ["running", "happiness", "flies", "fairly"]
stemmed = [stemmer.stem(word) for word in words]

print(stemmed)

# Output: ['run', 'happi', 'fli', 'fair']
```

Supported Languages

```
# List available languages
SnowballStemmer.languages
# ['arabic', 'danish', 'dutch', 'english', 'finnish', 'french', 'german', 'hungarian',
'italian', 'norwegian', 'porter', 'portuguese', 'romanian', 'russian', 'spanish', 'swedish']
```