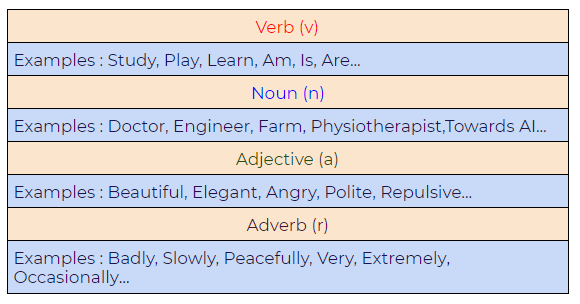
**Week 4: Lemmatization**

Lemmatization tries to achieve a similar base “stem” for a word. However, what makes it different is that it finds the dictionary word instead of truncating the original word. Stemming does not consider the context of the word. That is why it generates results faster, but it is less accurate than lemmatization.

If accuracy is not the project’s final goal, then stemming is an appropriate approach. If higher accuracy is crucial and the project is not on a tight deadline, then the best option is amortization (Lemmatization has a lower processing speed, compared to stemming).

Lemmatization takes into account Part Of Speech (POS) values. Also, lemmatization may generate different outputs for different values of POS. We generally have four choices for POS:

Figure 46: Part of Speech (POS) values in lemmatization.

Difference between Stemmer and Lemmatizer:

**a. Stemming:**

Notice how on stemming, the word “studies” gets truncated to “studi.”

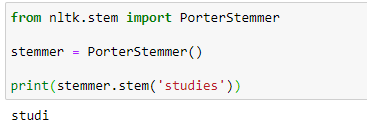


Figure 47: Using stemming with the NLTK Python framework.

**b. Lemmatizing:**

During lemmatization, the word “studies” displays its dictionary word “study.”

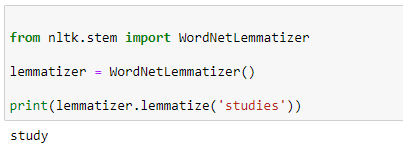


Figure 48: Using lemmatization with the NLTK Python framework.

Python Implementation:

**a. A basic example demonstrating how a lemmatizer works**

In the following example, we are taking the PoS tag as “verb,” and when we apply the lemmatization rules, it gives us dictionary words instead of truncating the original word:

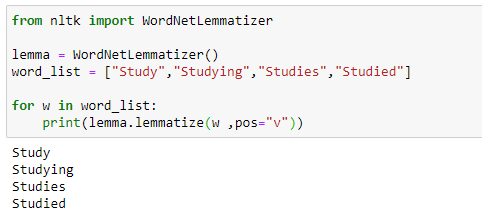


Figure 49: Simple lemmatization example with the NLTK framework.

**b. Lemmatizer with default PoS value**

The default value of PoS in lemmatization is a noun(n). In the following example, we can see that it’s generating dictionary words:

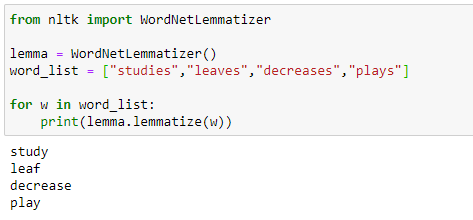


Figure 50: Using lemmatization to generate default values.

**c. Another example demonstrating the power of lemmatizer**

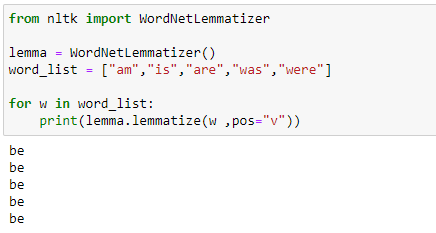


Figure 51: Lemmatization of the words: “am”, “are”, “is”, “was”, “were”

**d. Lemmatizer with different POS values**

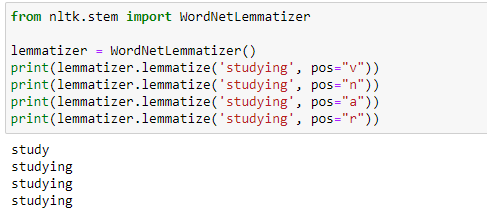


Figure 52: Lemmatization with different Part-of-Speech values

**Week 5: PoS Tagging with all PoS Tags**

**Why do we need Part of Speech (POS)?**

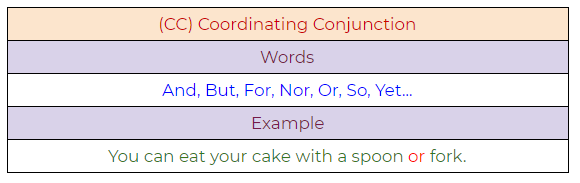
Figure 53: Sentence example, “can you help me with the can?”

Figure 53: Sentence example, “can you help me with the can?”

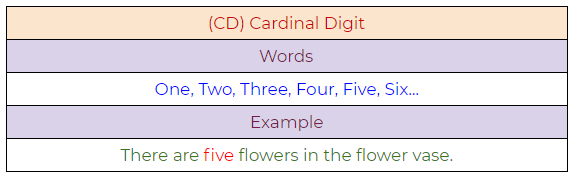
Parts of speech(PoS) tagging is crucial for syntactic and semantic analysis. Therefore, for something like the sentence above, the word “can” has several semantic meanings. The first “can” is used for question formation. The second “can” at the end of the sentence is used to represent a container. The first “can” is a verb, and the second “can” is a noun. Giving the word a specific meaning allows the program to handle it correctly in both semantic and syntactic analysis.

Below, please find a list of Part of Speech (PoS) tags with their respective examples:

**1. CC: Coordinating Conjunction**

Figure 54: Coordinating conjunction example.

**2. CD: Cardinal Digit**

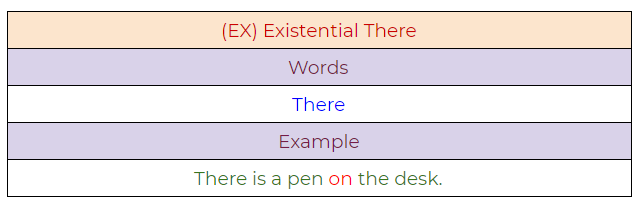
Figure 55: Cardinal digit example.

**3. DT: Determiner**

Graphical user interface, application

Description automatically generated with medium confidenceFigure 56: A determiner example.

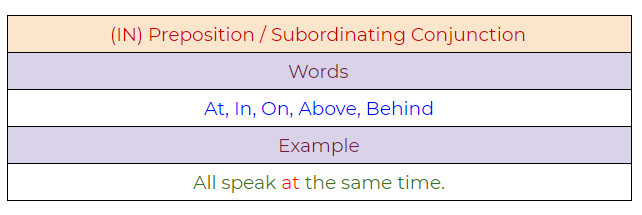
**4. EX: Existential There**

Figure 57: Existential “there” example.

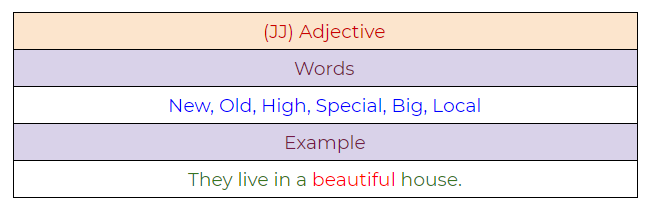
**5. FW: Foreign Word**

Figure 58: Foreign word example.

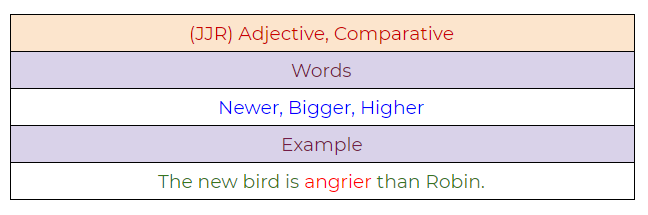
**6. IN: Preposition / Subordinating Conjunction**

Figure 59: Preposition/Subordinating conjunction.

**7. JJ: Adjective**

Figure 60: Adjective example.

**8. JJR: Adjective, Comparative**

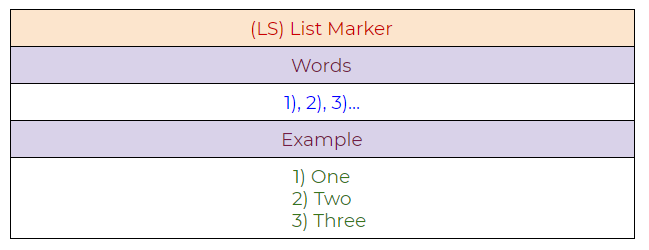
Figure 61: Adjective, comparative example.

**9. JJS: Adjective, Superlative**

Text, table

Description automatically generated with medium confidenceFigure 62:

**10. LS: List Marker**

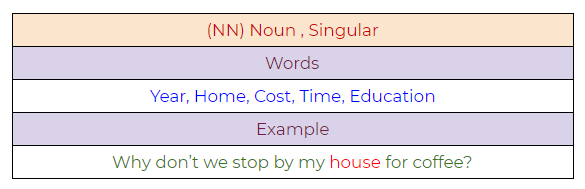
Figure 63: List marker example.

**11. MD: Modal**

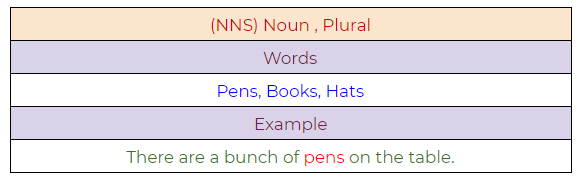
Table

Description automatically generated with medium confidenceFigure 64:

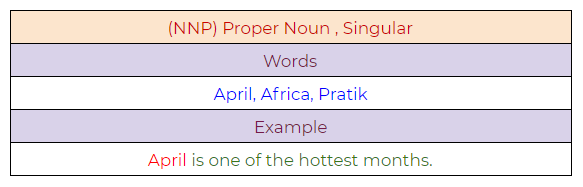
**12. NN: Noun, Singular**

Figure 65: Noun, singular example.

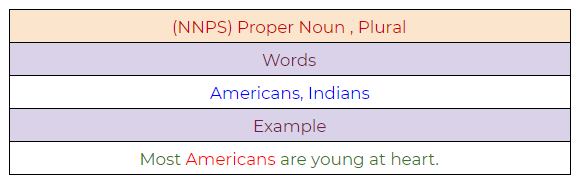
**13.**[**NNS**](https://towardsai.net/p/machine-learning/building-neural-networks-from-scratch-with-python-code-and-math-in-detail-i-536fae5d7bbf)**: Noun, Plural**

Figure 66: Noun, plural example.

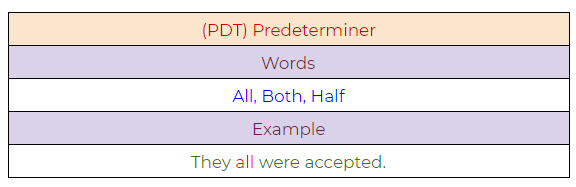
**14. NNP: Proper Noun, Singular**

Figure 67: Proper noun, singular example.

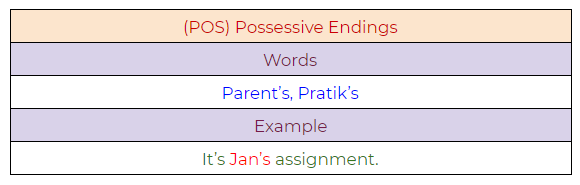
**15. NNPS: Proper Noun, Plural**

Figure 68: Proper noun, plural example.

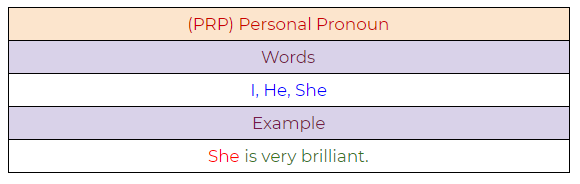
**16. PDT: Predeterminer**

Figure 69: Predeterminer example.

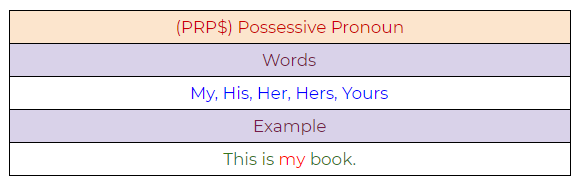
**17. POS: Possessive Endings**

Figure 70: Possessive endings example.

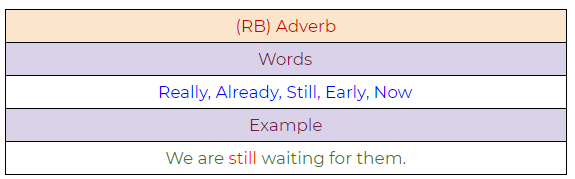
**18. PRP: Personal Pronoun**

Figure 71: Personal pronoun example.

**19. PRP$: Possessive Pronoun**

Figure 72: Possessive pronoun example.

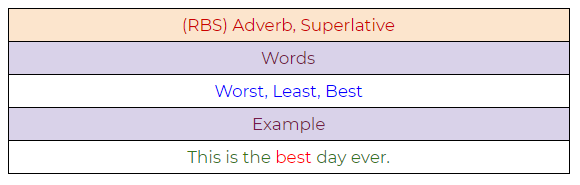
**20. RB: Adverb**

Figure 73: Adverb example.

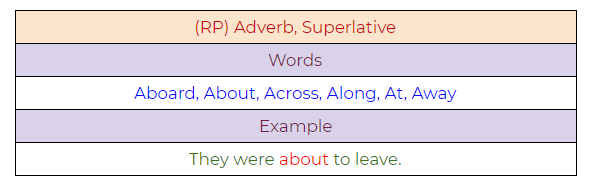
**21. RBR: Adverb, Comparative**

Figure 74: Adverb, comparative example.

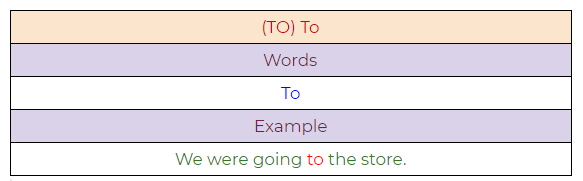
**22. RBS: Adverb, Superlative**

Figure 75: Adverb, superlative example.

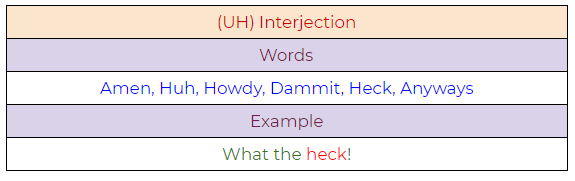
**23. RP: Particle**

Figure 76: Particle example.

**24. TO: To**

Figure 77: To example.

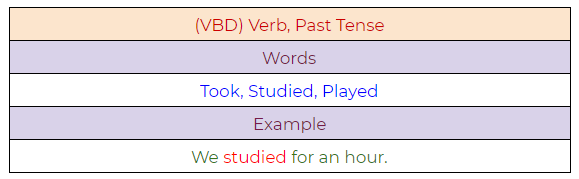
**25. UH: Interjection**

Figure 78: Interjection example.

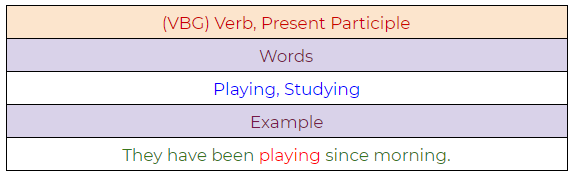
**26. VB: Verb, Base Form**

Figure 79: Verb, base form example.

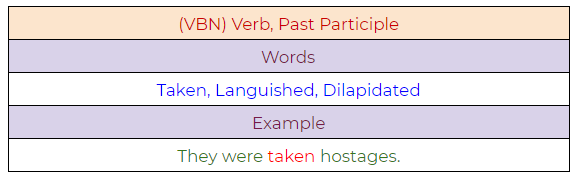
**27. VBD: Verb, Past Tense**

Figure 80: Verb, past tense example.

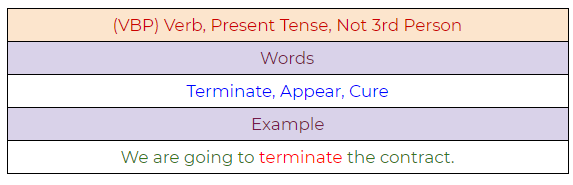
**28. VBG: Verb, Present Participle**

Figure 81: Verb, present participle example.

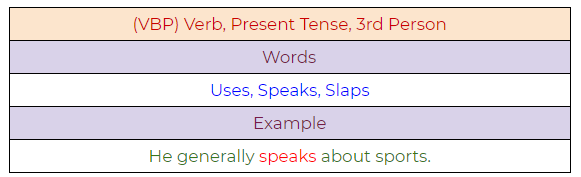
**29. VBN: Verb, Past Participle**

Figure 82: Verb, past participle.

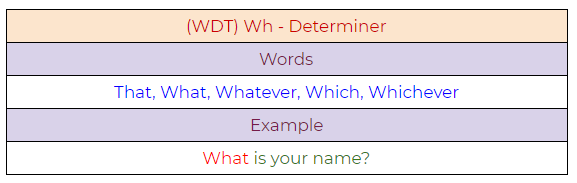
**30. VBP: Verb, Present Tense, Not Third Person Singular**

Figure 83: Verb, present tense, not third-person singular.

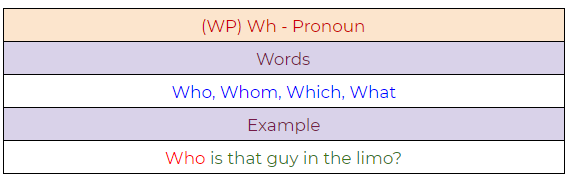
**31. VBZ: Verb, Present Tense, Third Person Singular**

Figure 84: Verb, present tense, third-person singular.

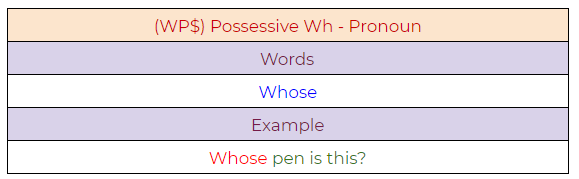
**32. WDT: Wh — Determiner**

Figure 85: Determiner example.

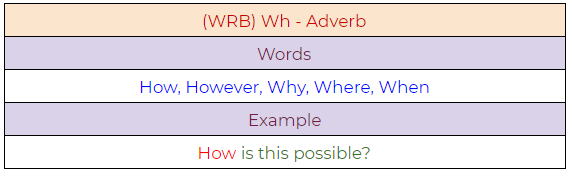
**33. WP: Wh — Pronoun**

Figure 86: Pronoun example.

**34. WP$ : Possessive Wh — Pronoun**

Figure 87: Possessive pronoun example.

**35. WRB: Wh — Adverb**

Figure 88: Adverb example.

Python Implementation:

**a. A simple example demonstrating PoS tagging.**

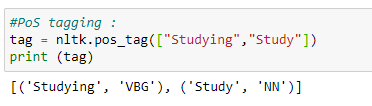


Figure 89: PoS tagging example.

**b. A full example demonstrating the use of PoS tagging.**

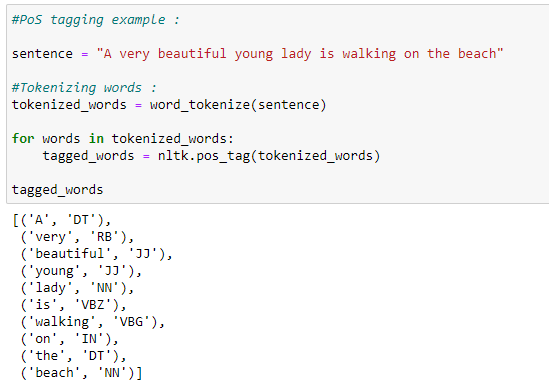


Figure 90: Full Python sample demonstrating PoS tagging.

**Week 6: Chunking Process**

Chunking means to extract meaningful phrases from unstructured text. By tokenizing a book into words, it’s sometimes hard to infer meaningful information. It works on top of Part of Speech(PoS) tagging. Chunking takes PoS tags as input and provides chunks as output. Chunking literally means a group of words, which breaks simple text into phrases that are more meaningful than individual words.

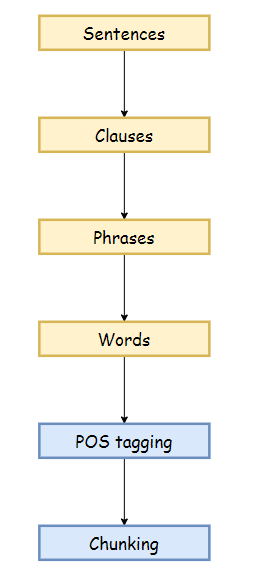


Figure 91: The chunking process in NLP.

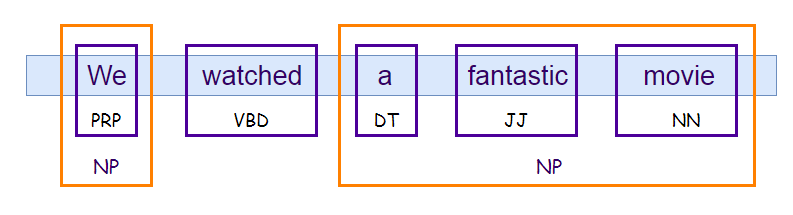
Before working with an example, we need to know what phrases are? Meaningful groups of words are called phrases. There are five significant categories of phrases.

1. Noun Phrases (NP).
2. Verb Phrases (VP).
3. Adjective Phrases (ADJP).
4. Adverb Phrases (ADVP).
5. Prepositional Phrases (PP).

Phrase structure rules:

* S(Sentence) → NP VP.
* NP → {Determiner, Noun, Pronoun, Proper name}.
* VP → V (NP)(PP)(Adverb).
* PP → Pronoun (NP).
* AP → Adjective (PP).

Example:

Figure 92: A chunking example in NLP.

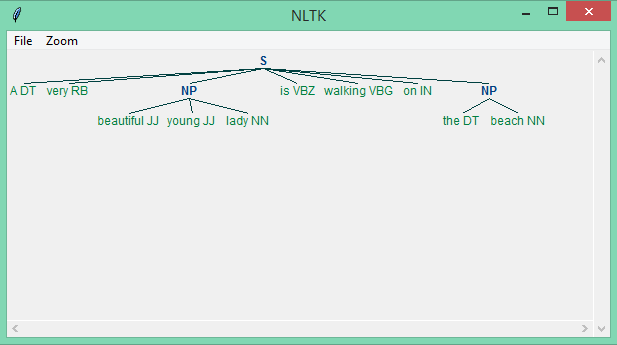
Python Implementation:

In the following example, we will extract a noun phrase from the text. Before extracting it, we need to define what kind of noun phrase we are looking for, or in other words, we have to set the grammar for a noun phrase. In this case, we define a noun phrase by an optional determiner followed by adjectives and nouns. Then we can define other rules to extract some other phrases. Next, we are going to use RegexpParser( ) to parse the grammar. Notice that we can also visualize the text with the .draw( ) function.



Figure 93: Code snippet to extract noun phrases from a text file.

In this example, we can see that we have successfully extracted the noun phrase from the text.

Figure 94: Successful extraction of the noun phrase from the input text.