Parts of Speech (PoS Tagging)

We cannot just tag any kind of tags, you cannot add random tags. For this, you need to follow some kind of convention. The standard convention for English language is PTB (Penn Treebank).

Suppose if you're giving a sentence as an input, (Hello, how are you?). The first step is tokenisation and then PoS tagging. Tags can be nouns, verbs, adjectives, adverbs, etc. Here the convention plays a crucial part. VV for verbs, NN for nouns (these are predefined and cannot be changed). This is called the naming convention. The problem with unknown language is that you don't have these standard tags and you'll have to create your own tags.

Types of POS:

1. Rule based POS: we define some predefined rules, for example there's a name, Aniket is a noun, Playing is a verb. so whenever you're giving some input like Abhay is playing, then according to the rule Abhay will be tagged with noun, is is a stopword, and playing is a verb. It heavily relies on the rules, kind of like machine learning

   https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html (https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html)

2. Transformation based tagging: Suppose you have a sentence (He is learning fast), we transform the sentence to make more meaning of the sentence

3. Statistical tagging : Conditional Random Fields & HMM. Here, we don't have any kind of rule, we mainly deal with the semantic of the language and based on the meaning, we get a probability based on the previous words of what word comes next. To get these probabilities, we need to go for it's semantic analysis.

Types of Taggers:

1. Default tagger: assigns a default tag, like NN for noun

2. Unigram: Uses training data to assign tags based on most likely tag.

3. Bigram: Considers the tag of the previous word and uses that info to assign a tag. It serves as a backup for unigram tagger. Preserve the meaning of the word.

In [1]:
```
import nltk
```

In [2]:
```
from nltk.tokenize import word_tokenize
from nltk.tag import pos_tag
from nltk.corpus import stopwords
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\ahana\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

Out[2]:
```
True
```

In [3]:
```
def simple_pos_tagging(text):
    tokens = word_tokenize(text)
    stop_words = set(stopwords.words('english'))
    filtered_tokens = [token for token in tokens if token.lower() not in stop_words]

    #part of speech tagging using nltk's pos_tag
    pos_tags = pos_tag(filtered_tokens)

    return pos_tags
```

In [4]:
```
#Example usage
text = "The boat was like a pea floating in a great bowl of blue soup"
result = simple_pos_tagging(text)
print("Parts of Speech Tags: ", result)
```

```
Parts of Speech Tags:  [('boat', 'NN'), ('like', 'IN'), ('pea', 'NN'), ('floating', 'VBG'), ('great', 'JJ'), ('bowl', 'NN'), ('blue',
'NN'), ('soup', 'NN')]
```

In [5]:
```
from nltk.tag import DefaultTagger, UnigramTagger, BigramTagger
```

In [6]:
```python
def compare_taggers(text):
    tokens = word_tokenize(text)
    stop_words = set(stopwords.words('english'))
    filtered_tokens = [token for token in tokens if token.lower() not in stop_words]

    #split data into training and testing sets
    train_size = int(0.8 * len(filtered_tokens))
    train_tokens, test_tokens = filtered_tokens[:train_size], filtered_tokens[train_size: ]

    #prepare tagged sentences for training
    tagged_sentences = [(token, 'NN') for token in train_tokens]

    #define tagging algorithms
    default_tagger = DefaultTagger('NN') #assings NN (noun) to all words
    unigram_tagger = UnigramTagger(model= {word:tag for word,tag in tagged_sentences})
    bigram_tagger = BigramTagger(train=[tagged_sentences], backoff=unigram_tagger)

    #evaluate performance on test set
    gold_standard = [(token, 'NN') for token in test_tokens]

    #NLTK expects a list of sentences, each sentence as a list of (word,token) tuples
    test_sents = [[(token,'NN') for token in test_tokens]]

    tagged_sents_default = default_tagger.tag_sents(test_sents)
    tagged_sents_unigram = unigram_tagger.tag_sents(test_sents)
    tagged_sents_bigram = bigram_tagger.tag_sents(test_sents)

    #extract tags for evaluation
    tags_default = [tag for (word,tag) in tagged_sents_default[0]]
    tags_unigram = [tag for (word,tag) in tagged_sents_unigram[0]]
    tags_bigram = [tag for (word,tag) in tagged_sents_bigram[0]]

    #calculate accuracy
    accuracy_default = sum(1 for gold, pred in zip(gold_standard, tags_default) if gold[1] == pred) / len(gold_standard)
    accuracy_unigram = sum(1 for gold, pred in zip(gold_standard, tags_unigram) if gold[1] == pred) / len(gold_standard)
    accuracy_bigram = sum(1 for gold, pred in zip(gold_standard, tags_bigram) if gold[1] == pred) / len(gold_standard)

    return accuracy_default, accuracy_unigram, accuracy_bigram

#Example usage
text = "The boat was like a pea floating in a great bowl of blue soup"
results = compare_taggers(text)
accuracy_default, accuracy_unigram, accuracy_bigram = compare_taggers(text)
print("Default Tagger Accuracy: ", accuracy_default)
print("Unigram Tagger Accuracy: ", accuracy_unigram)
print("Bigram Tagger Accuracy: ", accuracy_bigram)
```

```
Default Tagger Accuracy:  1.0
Unigram Tagger Accuracy:  0.0
Bigram Tagger Accuracy:  0.0
```

In [30]:
```python
#Assignment: Finetune the code to increase the accuracy of Unigram and Bigram Taggers.
```