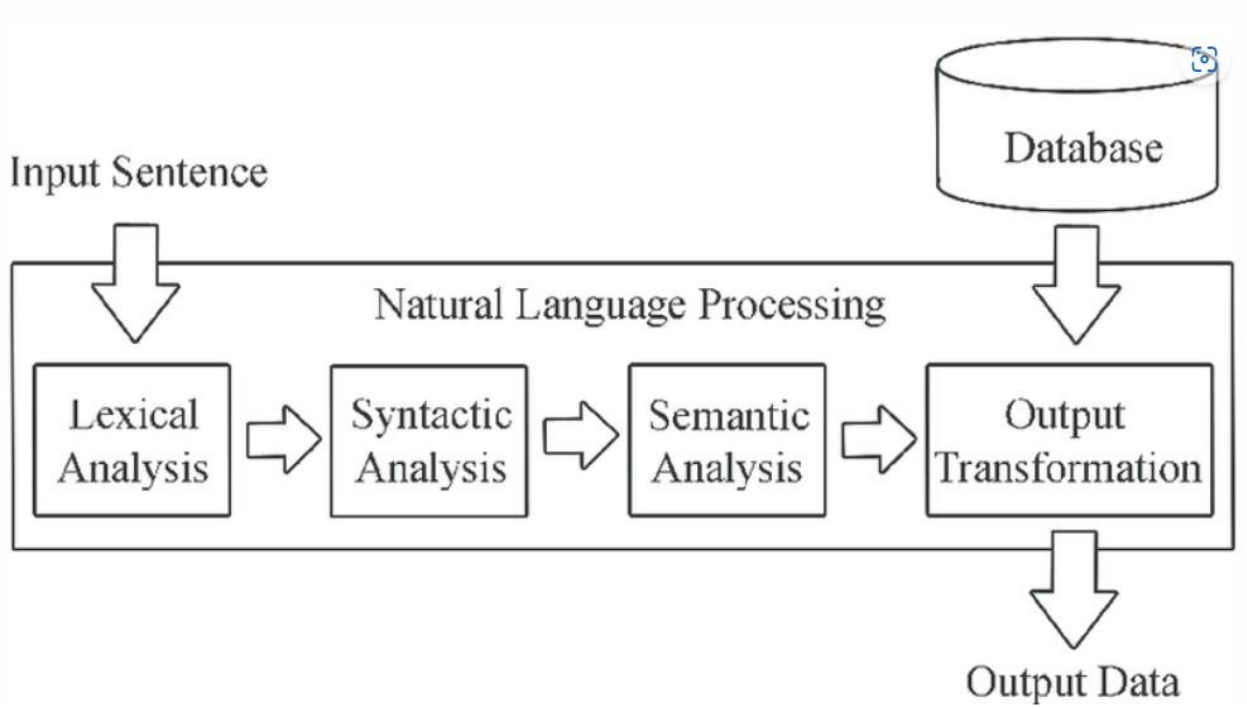
NATURAL LANGUAGE PROCESSING (DAY1)

Natural Language Processing (NLP) is a method within Artificial Intelligence (AI) that works with the analysis and building of intelligent systems that can function in languages that humans speak, for example, English. Processing of language is needed when a system wants to work based on input from a user in the form of text or speech and the user is adding input in regular use English.

NLP ACCOMPLISH:

- SPELLING/ GRAMMAR ACCURACY
- SPEECH RECOGNITION
- SMART SEARCH ENGINE
- MESSANGER BOTS OR CHATBOTS
- ASSITANTS (DIGITAL ASSISTANTS)



PARSING TEXT:

- LEXICAL ANALYSIS
- SYNTACTIC ANALYSIS
- SEMANTIC ANALYSIS

1. **LEXICAL ANALYSIS**: *To understand the* ***meaning*** **of words** *and note the* ***relationship*** **of one word to others**
   - ✓ Used as First step of complier
   - ✓ Breaks down the lines of code to a series of ' TOKENS ' (also prevents from white spaces)
   - ✓ Preserve multiple words together as an ' n-gram '
   - ✓ *Here both the TOKENS and N-GRAMS comes into the picture to understand the relations of words*

2. **SYNTACTIC ANALYSIS:** *Refers to the arrangement of words in a sentence so they* **GRAMMATICALLY MAKE SENSE.**
   - ✓ To check or apply grammar rules, a collection of algorithms is utilized to describe words and meaning from input
   - ✓ Syntax techniques are :-
     a) Lemmatization/ Stemming: Reduces word complexity to simpler forms that have less variation.
     - Strip away the suffix words or reduce the word to root word

     b) Parsing: Undergoing the grammatical analysis of a given sentence
     - DEPENDENCY PARSING, which assesses the relationships between words in a sentence

c) Word Segmentation: Separation of continuous text into separate words.
   - In English this easy because all the words are separated by spaces but for some languages like in the case of Japanese and Chinese they do not mark spaces for words
   - So the reason the Word Segmentation is very useful.

3. **SEMANTIC ANALYSIS:** *Refers to the* **MEANING THAT IS CONVEYED BY THE INPUT TEXT.**

- one of the most difficult task in natural language processing

✓ Involves NLP, requires algorithms to understand meaning, interpretation of words in addition to overall structure of a sentence. Techniques include:

a) Entity Extraction: Identifying and Extracting entities such as People, Place, Things etc..

b) Machine Translation: Automatically translate text from one to other language

c) Natural language Generation: Process of converting information of the computer semantic intention into readable human language.
   - Utilized by the chatbots to effectively and realistically respond to the users
   - Examples: Google Dailogflow, IBM Watson etc....

d) Natural Language Understanding: Involves converting pieces of text into representations that are structured logically for the computer programs to easily manipulate

- NATURAL LANGUAGE GENERATION IS UTILIZED BY CHATBOTS (important point)

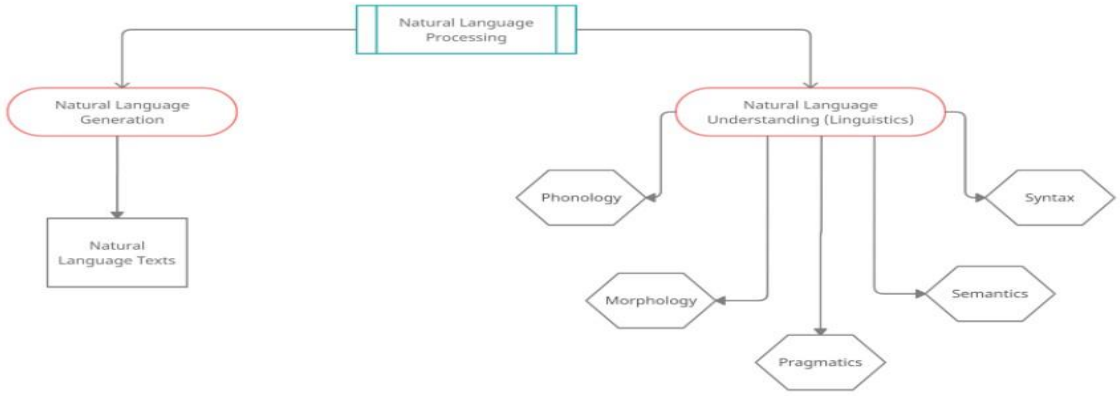
NATURAL LANGUAGE PROCESSING (DAY2)----- PART1

Challenges in Natural Language Processing

1. **Lexical Ambiguity:** This is the first level of ambiguity that occurs generally in words alone. For instance, when a code is given a word like 'board' it would not know whether to take it as a noun or a verb. This causes ambiguity in the processing of this piece of code.

2. **Syntax Level Ambiguity:** This is another type of ambiguity that has more to do with the way phrases sound in comparison to how the machine perceives it. For instance, a sentence like, 'He raised the scuttle with a blue cap'. This could mean one of two things. Either he raised a scuttle with the help of a blue cap, or he raised a scuttle that had a red cap.

3. **Referential Ambiguity:** References made using pronouns constitute referential ambiguity. For instance, two girls are running on the track. Suddenly, she says, 'I am exhausted'. It is not possible for the program to interpret, who out of the two girls is tired.



NLP Challenges

While extremely powerful, NLP models can also be difficult to build because of the dynamic and nuanced nature of human languages. Some of these challenges include:

- Human language is extremely dynamic. The meaning of words changes subtly over time, and new words are constantly introduced into use. This means that NLP models must follow word trends, and understand how those tie into concepts and messages.
- Human lanugage is nuanced. The complex characteristics of human languages such as sarcasm and suffixes cause problems for NLP. High level emotive constructs, like sarcasm, are subtle and abstract for a machine to pick up on. Low-leve problems like suffixes can be a bit easier for a machine to decipher, but still present difficulties as the machine may confuse variations of one word with contractions or endings of another.

Even with these challenges, there are many powerful computer algorithms that can be used to extract and structure from text.

NATURAL LANGUAGE PROCESSING (DAY2) -----PART 2

PARSING:

- Parsing is the process of analyzing the sentence for its structure, content and meaning, i.e. to uncover the structure, articulate the constituents and the relation between the constituents of the input sentence

PARSING IN NLP:

- Used to **DRAWN THE EXACT MEANING FROM THE TEXT**

- **PARSING IS THE PRIME TASK IN PROCESSING OF NATURAL LANGUAGE PROCESSING**, as it forms the basis for all the natural language processing applications, like machine translation, question answering and information retrieval.

- Also known as **SYNTAX** or **SYNTACTIC ANALYSIS** (which means grammar analysis from input or given sentence)

- In simple terms **PARSING IS BREAKING DOWN OF SENTENCE INTO ATOMIC VALUES**. "To analyze data or a sentence for structure, content and meaning in terms of grammatical constituents, identifying the parts of speech, syntactic relations".

For example in the sentence, "John is playing game" part-of-speech for each token is "noun" for "John" and "game", "verb" for "is" and "playing".

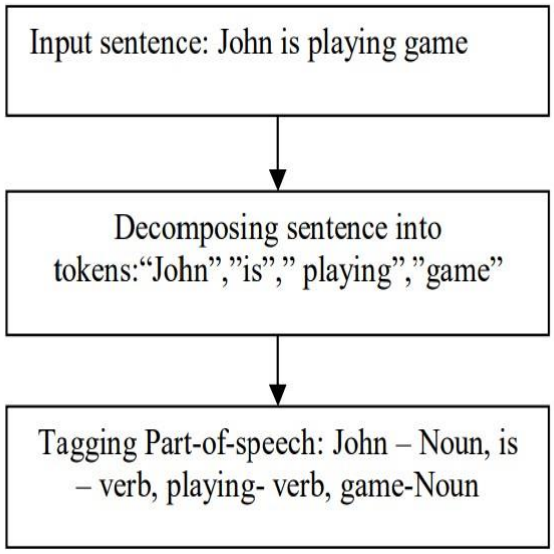
Making a **DISAMBIGUOUS** decision means, finding the correct part-of-speech for a word having multiple part-of-speeches, which give rise to **"AMBIGUITY"**.

- **AMBIGUITY** means having more than one interpretation of word or sentence.

Example “book”, it can be “noun” or “verb”, depending upon its use, parsing is use to find the correct parse for a word or a sentence.

PARSE TREE

- Parsing results in **GENERATION OF PARSE TREE**
- Which is the **GRAPHICAL REPRESENTATION OF THE ORDER** in which **THE GRAMMAR PRODUCTIONS ARE APPLIED DURING PARSING** of a sentence.
- Therefore parsing can be viewed as the order in which the nodes of parse tree are constructed.
- Describes how the grammar was used to produce the sentence



Parsing process

RELEVANCE OF PARSING IN NATURAL LANGUAGE PROCESSING:

- Parser is used to report any syntax error
- Parse tree is created with help of parser
- Create symbol table, which plays an important role in NLP
- Parser is also used to produce intermediate representations [IR]

DEEP PARSING AND SHALLOW PARSING

DEEP PARSING	SHALLOW PARSING
Which is also known as FULL PARSING	Which is known as CHUNKING
Search strategy will give a complete syntactic structure to a sentence	Search strategy is to limited part of the syntactic information from the given task
Complex NLP applications	Less complex information
Applications like: Dialogue systems and Summarizations	Applications like: Information extraction, Text mining

PARSING TECHNIQUES

- TOP DOWN PARSING
- BOTTOM UP PARSING
- RECURSIVE PARSING
- LR PARSER

NATURAL LANGUAGE TOOL KIT (NLTK) CONTAINS TWO WAY RO DO DEPENDENCY PARSER:

- **PROBABILISTIC, PROJECTIVE DEPENDENCY PARSER**

and

- **STANFORD PARSER**

Define *PARSING* or *SYNTACTIC PARSING* in Natural Language Processing??

- The process of analyzing the strings of symbols in natural language conforming to the rules of formal grammar.

NATURAL LANGUAGE PROCESSING (DAY3)

VARIOUS TYPES OF PARSERS

As discussed, a parser is basically a procedural interpretation of grammar. It finds an optimal tree for the given sentence after searching through the space of a variety of trees. Let us see some of the available parsers below −

Recursive descent parser

Recursive descent parsing is one of the most straightforward forms of parsing. Following are some important points about recursive descent parser −

- It follows a top down process.
- It attempts to verify that the syntax of the input stream is correct or not.
- It reads the input sentence from left to right.
- One necessary operation for recursive descent parser is to read characters from the input stream and matching them with the terminals from the grammar.

Shift-reduce parser

Following are some important points about shift-reduce parser −

- It follows a simple bottom-up process.
- It tries to find a sequence of words and phrases that correspond to the right-hand side of a grammar production and replaces them with the left-hand side of the production.
- The above attempt to find a sequence of word continues until the whole sentence is reduced.
- In other simple words, shift-reduce parser starts with the input symbol and tries to construct the parser tree up to the start symbol.

Chart parser

Following are some important points about chart parser −

- It is mainly useful or suitable for ambiguous grammars, including grammars of natural languages.
- It applies dynamic programing to the parsing problems.
- Because of dynamic programing, partial hypothesized results are stored in a structure called a ‘chart’.
- The ‘chart’ can also be re-used.

Regexp parser

Regexp parsing is one of the mostly used parsing technique. Following are some important points about Regexp parser −

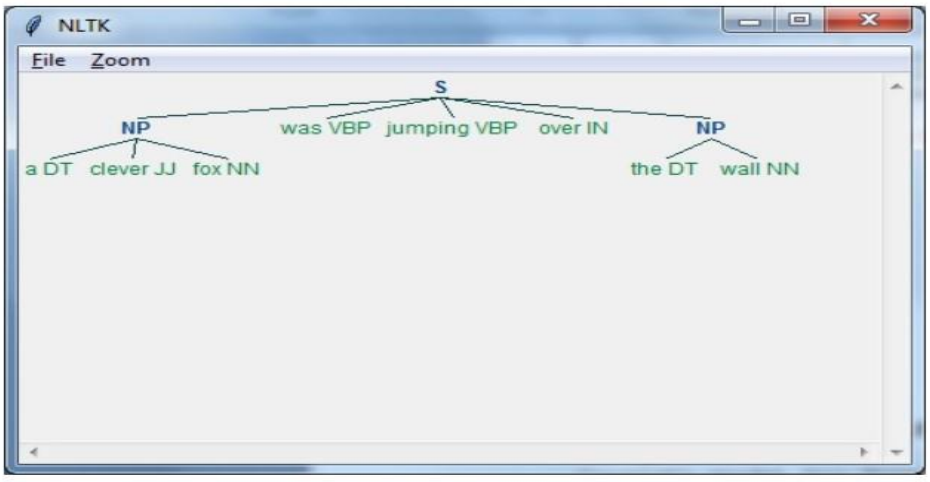
- As the name implies, it uses a regular expression defined in the form of grammar on top of a POS-tagged string.
- It basically uses these regular expressions to parse the input sentences and generate a parse tree out of this.

Example

Following is a working example of Regexp Parser −

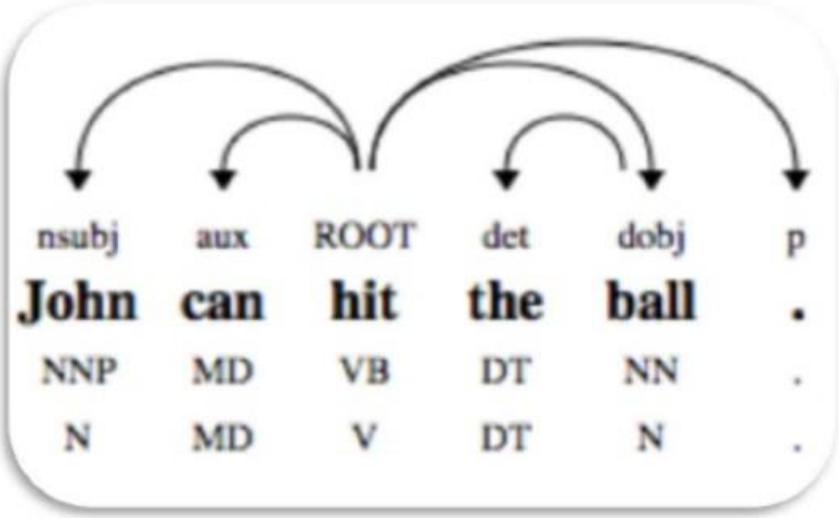
```
import nltk
sentence = [
   ("a", "DT"),
   ("clever", "JJ"),
   ("fox","NN"),
   ("was","VBP"),
   ("jumping","VBP"),
   ("over","IN"),
   ("the","DT"),
   ("wall","NN")
]
grammar = "NP:{<DT>?<JJ>*<NN>}"
Reg_parser = nltk.RegexpParser(grammar)
Reg_parser.parse(sentence)
Output = Reg_parser.parse(sentence)
Output.draw()
```

Output



Dependency Parsing

Dependency Parsing (DP), a modern parsing mechanism, whose main concept is that each linguistic unit i.e. words relates to each other by a direct link. These direct links are actually ***'dependencies'*** in linguistic. For example, the following diagram shows dependency grammar for the sentence ***"John can hit the ball"***.



NLTK Package

We have following the two ways to do dependency parsing with NLTK −

Probabilistic, projective dependency parser

This is the first way we can do dependency parsing with NLTK. But this parser has the restriction of training with a limited set of training data.

Stanford parser

This is another way we can do dependency parsing with NLTK. Stanford parser is a state-of-the-art dependency parser. NLTK has a wrapper around it. To use it we need to download following two things −

The Stanford CoreNLP parser.

Language model for desired language. For example, English language model.

Example

Once you downloaded the model, we can use it through NLTK as follows −

```
from nltk.parse.stanford import StanfordDependencyParser
path_jar = 'path_to/stanford-parser-full-2014-08-27/stanford-parser.jar'
path_models_jar = 'path_to/stanford-parser-full-2014-08-27/stanford-parser-3.4.1-model
dep_parser = StanfordDependencyParser(
   path_to_jar = path_jar, path_to_models_jar = path_models_jar
)
result = dep_parser.raw_parse('I shot an elephant in my sleep')
depndency = result.next()
list(dependency.triples())
```

Output

```
[
   ((u'shot', u'VBD'), u'nsubj', (u'I', u'PRP')),
   ((u'shot', u'VBD'), u'dobj', (u'elephant', u'NN')),
   ((u'elephant', u'NN'), u'det', (u'an', u'DT')),
   ((u'shot', u'VBD'), u'prep', (u'in', u'IN')),
   ((u'in', u'IN'), u'pobj', (u'sleep', u'NN')),
   ((u'sleep', u'NN'), u'poss', (u'my', u'PRP$'))
]
```

NATURAL LANGUAGE PROCESSING (DAY4)

Preprocessing of TEXT in NLP:

- The input in natural language processing is **TEXT**. The data collection for this text happens from a lot of sources. This requires a lot of **CLEANING** and **PROCESSING** before the data can be used for analysis.

- First thing you need to do in any NLP project is **TEXT PREPROCESSING**. Preprocessing input text simply means putting the data into a predictable and analyzable form. It's a crucial step for building an amazing NLP application.

These are some of the methods of processing the data in NLP:

- Tokenization
- Stop words removal
- Stemming
- Normalization
- Lemmatization
- Parts of speech tagging

Tokenization

Tokenization is **BREAKING THE RAW TEXT INTO SMALL CHUNKS**.

Tokenization breaks the raw text into words, sentences called **TOKENS**. These tokens help in understanding the context or developing the model for the NLP. The tokenization helps in interpreting the meaning of the text by analyzing the sequence of the words.

NEED OF TOKENIZATION

- First step in any NLP pipeline. A tokenizer breaks unstructured data and natural language text into chunks of information that can be used directly as **VECTOR REPRESENTING THAT DOCUMENT**.
- This transforming from unstructured string (text document) into numerical data structure suitable for machine learning. They can also **USE DIRECTLY BY A COMPUTER TO TRIGGER USEFUL ACTIONS AND RESPONSES**.

Different methods and libraries for tokenization

We can perform TOKENIZATION using different methods and techniques:

- **NLTK**
- **GENSIM**
- **KERAS**
- **SPACY** etc…

EXAMPLES OF TOKENIZATIONS:

1. "Tokenization can be done to either separate words or sentences. If the text is split into words using some separation technique they it is called as WORD TOKENIZATION."

SENTENCE TOKENIZATION METHOD:

----" Tokenization can be done to either separate words or sentences"

"If the text is split into words using some separation technique they it is called as Word Tokenization"

2. “When separation technique is used on sentences they it is known as Sentence Tokenization”

WORD TOKENIZATION METHOD:

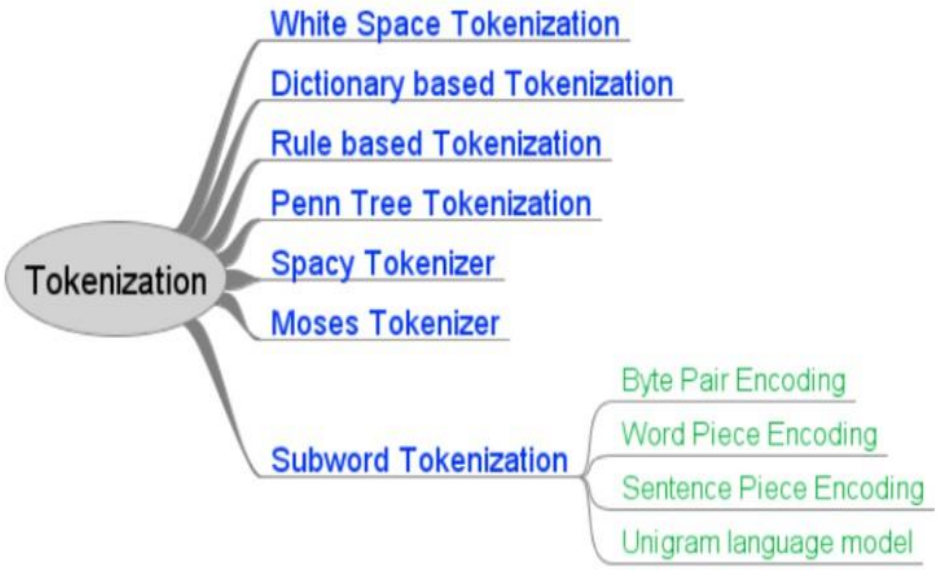
-------“ when”, ”separation”, ”technique”, ”used”, ”sentences”, ”known”, ”sentence”, ”tokenization”

From the above *Word Tokenization Method* example, we found some of the words are missed after the tokenization. Those kind of words are known **STOP WORDS**.

Stop words are those words in the text which does not add any meaning to the sentence and their removal will not affect the processing of text for the defined purpose. They are removed from the vocabulary to reduce noise and to reduce the dimension of the feature set.

Tokenization can be done to either separate words or sentences. If the text is split into words using some separation technique it is called word tokenization and same separation done for sentences is called sentence tokenization.

DIFFERENT TOOLS FOR TOKENIZATION



White Space Tokenization

Given a sentence or paragraph it tokenizes into words by splitting the input whenever a white space in encountered. This is the fastest tokenization technique but will work for languages in which the white space breaks apart the sentence into meaningful words. Example: English language.

Examples:

```
sentence = "I was born in Tunisia in 1995."
sentence.split()
```

```
['I', 'was', 'born', 'in', 'Tunisia', 'in', '1995.']
```

```
sentence = "I was born in Tunisia in 1995, I am 26 years old"
sentence.split(', ')
```

```
['I was born in Tunisia in 1995', 'I am 26 years old']
```

Dictionary Based Tokenization

In this method the tokens are found based on the tokens already existing in the dictionary. If the token is not found, then special rules are used to tokenize it. It is an advanced technique compared to whitespace tokenizer.

Rule Based Tokenization

In this technique a set of rules are created for the specific problem. The tokenization is done based on the rules. For example creating rules bases on grammar for particular language.

Regular Expression Tokenizer

This technique uses regular expression to control the tokenization of text into tokens. Regular expression can be simple to complex and sometimes difficult to comprehend. This technique should be preferred when the above methods does not serve the required purpose. It is a rule based tokenizer.

Example:

```
import re
text = """Founded in 2002, SpaceX’s mission is to enable humans to become a spacefaring civilization and a multi-planet
species by building a self-sustaining city on, Mars. In 2008, SpaceX’s Falcon 1 became the first privately developed
liquid-fuel launch vehicle to orbit the Earth."""
sentences = re.compile('[.!?] ').split(text)
sentences
```

```
Output : ['Founded in 2002, SpaceX’s mission is to enable humans to become a spacefaring
           civilization and a multi-planet \nspecies by building a self-sustaining city on
           Mars.',
          'In 2008, SpaceX’s Falcon 1 became the first privately developed \nliquid-fuel
           launch vehicle to orbit the Earth.']
```

Penn TreeBank Tokenization

Tree bank is a corpus created which gives the semantic and syntactical annotation of language. Penn Treebank is one of the largest treebanks which was published. This technique of tokenization separates the punctuation, clitics (words that occur along with other words like I’m, don’t) and hyphenated words together.

Example:

```
text="What you don't want to be done to yourself, don't do to others..."
tokenizer= TreebankWordTokenizer()
print(tokenizer.tokenize(text))
```

```
['What', 'you', 'do', "n't", 'want', 'to', 'be', 'done', 'to', 'yourself', ',', 'do', "n't", 'do', 'to', 'others',
'...']
```

Spacy Tokenizer

This is a modern technique of tokenization which faster and easily customizable. It provides the flexibility to specify special tokens that need not be segmented or need to be segmented using special rules. Suppose you want to keep $ as a separate token, it takes precedence over other tokenization operations.

```
text= "All happy families are alike; each unhappy family is unhappy in its own way!!!👌👌 #Leo Tolstoy "
doc = nlp(text)
for token in doc:
    print(token, token.idx)
```

```
All 0
happy 4
families 10
are 19
alike 23
; 28
each 30
unhappy 35
family 43
is 50
unhappy 53
in 61
its 64
own 68
way 72
! 75
! 76
! 77
👌 78
👌 79
# 81
Leo 82
Tolstoy 86
```

Moses Tokenizer

This is a tokenizer which is advanced and is available before Spacy was introduced. It is basically a collection of complex normalization and segmentation logic which works very well for structured language like English.

Subword Tokenization

This tokenization is very useful for specific application where sub words make significance. In this technique the most frequently used words are given unique ids and less frequent words are split into sub words and they best represent the meaning independently. For example if the word few is appearing frequently in the text it will be assigned a unique id, where fewer and fewest which are rare words and are less frequent in the text will be split into sub words like few, er, and est. This helps the language model not to learn fewer and fewest as two separate words. This allows to identify the unknown words in the data set during training. There are different types of subword tokenization and they are given below and Byte-Pair Encoding and WordPiece will be discussed briefly.

- Byte-Pair Encoding (BPE)
- WordPiece
- Unigram Language Model
- SentencePiece

Byte-Pair Encoding (BPE)

This technique is based on the concepts in information theory and compression. BPE uses Huffman encoding for tokenization meaning it uses more embedding or symbols for representing less frequent words and less symbols or embedding for more frequently used words.

The BPE tokenization is bottom up sub word tokenization technique. The steps involved in BPE algorithm is given below.

1. Starts with splitting the input words into single unicode characters and each of them corresponds to a symbol in the final vocabulary.
2. Find the most frequent occurring pair of symbols from the current vocabulary.
3. Add this to the vocabulary and size of vocabulary increases by one.
4. Repeat steps ii and iii till the defined number of tokens are built or no new combination of symbols exist with required frequency.

WordPiece

WordPiece is similar to BPE techniques expect the way the new token is added to the vocabulary. BPE considers the token with most frequent occurring pair of symbols to merge into the vocabulary. While WordPiece considers the frequency of individual symbols also and based on below count it merges into the vocabulary.

Count (x, y) = frequency of (x, y) / frequency (x) * frequency (y)

The pair of symbols with maximum count will be considered to merge into vocabulary. So it allows rare tokens to be included into vocabulary as compared to BPE.