In [66]:
```
from nltk.tokenize import WhitespaceTokenizer
from nltk.tokenize import WordPunctTokenizer
from nltk.tokenize import TreebankWordTokenizer
from nltk.tokenize import TweetTokenizer
from nltk.tokenize import MWETokenizer
```

In [67]:
```
sent = "All Indians are ..my brothers and I don't have sisters. I love my country and I am proud of its rich and varie
```

In [68]:
```
sent
```

Out[68]:
```
"All Indians are ..my brothers and I don't have sisters. I love my country and I am proud of its rich and varied heri
tage. I shall strive to be worthy !~!!of it. I --shall respect my parents, teachers and all elders and treat everyone
with courtesy😀😀😀😀😀😀😃😃🤣🤣."
```

TOKENIZATION

Tokenization is the first step in any NLP pipeline. It has an important effect on the rest of your pipeline. A tokenizer breaks unstructured data and natural language text into chunks of information that can be considered as discrete elements. The token occurrences in a document can be used directly as a vector representing that document.

White Space Tokenization

The simplest way to tokenize text is to use whitespace within a string as the “delimiter” of words. This can be accomplished with Python’s split function, which is available on all string object instances as well as on the string built-in class itself. You can change the separator any way you need.

In [69]:
```
whitespace_token = WhitespaceTokenizer().tokenize(sent)
print(whitespace_token,"\nlength of tokens:",len(whitespace_token))
```

```
['All', 'Indians', 'are', '..my', 'brothers', 'and', 'I', "don't", 'have', 'sisters.', 'I', 'love', 'my', 'country',
'and', 'I', 'am', 'proud', 'of', 'its', 'rich', 'and', 'varied', 'heritage.', 'I', 'shall', 'strive', 'to', 'be', 'wo
rthy', '!~!!of', 'it.', 'I', '--shall', 'respect', 'my', 'parents,', 'teachers', 'and', 'all', 'elders', 'and', 'trea
t', 'everyone', 'with', 'courtesy😀😀😀😀😀😀😃😃🤣🤣.']
length of tokens: 46
```

Punctuation-based tokenizer

This tokenizer splits the sentences into words based on whitespaces and punctuations.

In [70]:
```
punct_based =WordPunctTokenizer().tokenize(sent)
print(punct_based,"\n Length of tokens:",len(punct_based))
```

```
['All', 'Indians', 'are', '..', 'my', 'brothers', 'and', 'I', 'don', "'", 't', 'have', 'sisters', '.', 'I', 'love',
'my', 'country', 'and', 'I', 'am', 'proud', 'of', 'its', 'rich', 'and', 'varied', 'heritage', '.', 'I', 'shall', 'str
ive', 'to', 'be', 'worthy', '!~!!', 'of', 'it', '.', 'I', '--', 'shall', 'respect', 'my', 'parents', ',', 'teachers',
'and', 'all', 'elders', 'and', 'treat', 'everyone', 'with', 'courtesy', '😀😀😀😀😀😀😃😃🤣🤣.']
 Length of tokens: 56
```

Treebank Word tokenizer

This tokenizer incorporates a variety of common rules for english word tokenization. It separates phrase-terminating punctuation like (?!.;,) from adjacent tokens and retains decimal numbers as a single token. Besides, it contains rules for English contractions.

For example “don’t” is tokenized as [“do”, “n’t”]. You can find all the rules for the Treebank Tokenizer at this link. https://www.nltk.org/api/nltk.tokenize.html#module-nltk.tokenize.treebank (https://www.nltk.org/api/nltk.tokenize.html#module-nltk.tokenize.treebank)

In [71]:
```
tree_bank =TreebankWordTokenizer().tokenize(sent)
print(tree_bank,"\nLength of tokens:",len(tree_bank))
```

```
['All', 'Indians', 'are', '..my', 'brothers', 'and', 'I', 'do', "n't", 'have', 'sisters.', 'I', 'love', 'my', 'countr
y', 'and', 'I', 'am', 'proud', 'of', 'its', 'rich', 'and', 'varied', 'heritage.', 'I', 'shall', 'strive', 'to', 'be',
'worthy', '!', '~', '!', '!', 'of', 'it.', 'I', '--', 'shall', 'respect', 'my', 'parents', ',', 'teachers', 'and', 'a
ll', 'elders', 'and', 'treat', 'everyone', 'with', 'courtesy😀😀😀😀😀😀😃😃🤣🤣', '.']
Length of tokens: 54
```

Tweet tokenizer

When we want to apply tokenization in text data like tweets, the tokenizers mentioned above can’t produce practical tokens. Through this issue, NLTK has a rule based tokenizer special for tweets. We can split emojis into different words if we need them for tasks like sentiment analysis.

In [72]:
```
tweet = TweetTokenizer().tokenize(sent)
print(tweet,"\nlength of tokens:",len(tweet))
```

```
['All', 'Indians', 'are', '..', 'my', 'brothers', 'and', 'I', "don't", 'have', 'sisters', '.', 'I', 'love', 'my', 'co
untry', 'and', 'I', 'am', 'proud', 'of', 'its', 'rich', 'and', 'varied', 'heritage', '.', 'I', 'shall', 'strive', 't
o', 'be', 'worthy', '!', '~', '!', '!', 'of', 'it', '.', 'I', '-', '-', 'shall', 'respect', 'my', 'parents', ',', 'te
achers', 'and', 'all', 'elders', 'and', 'treat', 'everyone', 'with', 'courtesy', '😀', '😀', '😀', '😃', '😃',
'🤣', '🤣', '.']
length of tokens: 65
```

MWET tokenizer

NLTK's multi-word expression tokenizer (MWETokenizer) provides a function add_mwe() that allows the user to enter multiple word expressions before using the tokenizer on the text. More simply, it can merge multi-word expressions into single tokens.

In [78]:
```
sent1= "The Hunger Games film series is composed of science fiction dystopian adventure films, based on The Hunger Gam
mwe=MWETokenizer()
mwe.add_mwe(('Hunger','Games'))
mwe1=mwe.tokenize(sent1.split())
print(mwe1,"\nLength of tokens:",len(mwe1))
```

```
['The', 'Hunger_Games', 'film', 'series', 'is', 'composed', 'of', 'science', 'fiction', 'dystopian', 'adventure', 'fi
lms,', 'based', 'on', 'The', 'Hunger_Games', 'trilogy', 'of', 'novels', 'by', 'American', 'author', 'Suzanne', 'Colli
ns.']
Length of tokens: 24
```

In [ ]:
```

```