Name: Ranjani **Reg no: 19MAI0004**

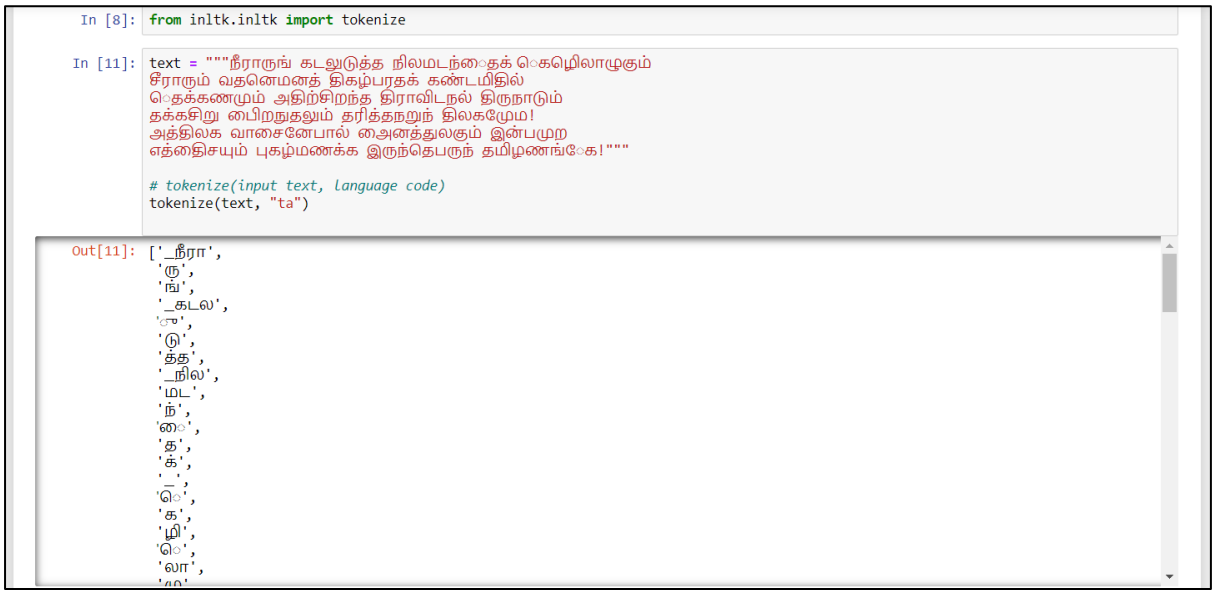
Tasks-1

1. 5 different functions of nlp in our Regional Language (TAMIL)

This is the small text I used which I wrote in Transliteration

Nīrāruṅ kaṭaluṭutta nilamaṭanaitak ̆ekaḷielāḷukum
cīrārum vataṉemaṉat tikaḷparatak kaṇṭamitil
etakkaṇamum atiṟciṟanta tirāviṭanal tirunāṭum
takkaciṟu piaiṟanutalum tarittanaṟun tilakamuēma!
Attilaka vācaiṉēpāl ȧaiṉattulakum iṉpamuṟa
ettiaicayum pukaḷmaṇakka irunteparun tamiḻaṇaṅēka!

I Tokenize



Here we took some text in Tamil and already installed and imported the inltk for Indian languages and just tokenized them.

II Get embedding vectors

Getting embedding vectors

```
In [4]: from inltk.inltk import get_embedding_vectors

        #vectors = get_embedding_vectors(text, 'ta') # where text is string in <code-of-language>


        vectors = get_embedding_vectors('நீராருங் ', 'ta')
        vectors[0].shape
```

```
Out[4]: (400,)
```

```
In [6]: get_embedding_vectors('சீராரும் வதனெமனத் திகழ்பரதக் கண்டமிதில்','ta')
```

```
Out[6]: [array([-0.392376, -0.026561,  0.218991, -0.613945, ..., -0.468301,  0.088901,  0.606461,  0.141723], dtype=float32),
         array([ 0.241516, -0.136911,  0.356693, -0.873655, ..., -0.540106, -0.846683, -1.429399, -0.03362 ], dtype=float32),
         array([-0.121189, -0.060486,  0.203301,  0.097394, ..., -0.334376, -0.726214, -1.246265,  0.345443], dtype=float32),
         array([-0.268574,  0.578104, -0.092597, -0.100336, ..., -0.491912, -1.095734,  0.5956  ,  0.268935], dtype=float32),
         array([ 0.03303 , -0.013981, -0.955797, -0.016221, ..., -0.827908, -1.914278,  0.650999,  0.350428], dtype=float32),
         array([-0.035348,  0.201428,  0.392425, -0.107964, ..., -0.6994  , -0.521678, -0.437072, -0.293238], dtype=float32),
         array([-0.450861, -0.374799,  0.120025,  0.340392, ..., -0.403203,  0.117812,  0.70024 ,  0.504618], dtype=float32),
         array([ 0.090684, -0.043384,  0.830132,  0.777868, ..., -0.609601, -0.942782, -0.367691, -0.124289], dtype=float32),
         array([-0.54909 ,  0.03791 , -0.038977,  0.149942, ..., -0.781892,  0.639784, -0.06479 ,  0.424908], dtype=float32),
         array([-0.880802,  0.167188, -0.819308,  0.433803, ..., -0.616377, -0.29351 , -1.89064 ,  0.189198], dtype=float32),
         array([ 0.027565,  0.197675,  0.390107, -0.210334, ..., -0.557   ,  0.030082, -0.16497 ,  0.798928], dtype=float32),
         array([ 0.03303 , -0.013981, -0.955797, -0.016221, ..., -0.827908, -1.914278,  0.650999,  0.350428], dtype=float32),
         array([ 0.292957,  0.038081,  0.542439,  0.421605, ..., -0.532235, -0.942552,  0.462055,  0.091556], dtype=float32),
         array([ 0.434617,  0.312698,  0.029044, -0.120213, ..., -0.333071, -0.068938,  0.176448,  1.632599], dtype=float32),
         array([ 7.404844e-04,  7.676391e-02,  3.397048e-01, -5.299678e-01, ..., -4.810513e-01, -6.573264e-01, -2.172701e-01,
                -8.429354e-01], dtype=float32),
         array([-0.353654, -0.127774, -0.305252,  0.385747, ..., -0.676172,  0.130398,  0.08781 , -0.449366], dtype=float32)]
```

```
In [7]: len(vectors)
```

```
Out[7]: 3
```

This returns an array of “Embedding vectors”, containing 400-Dimensional representation for every token in the text.

III Predict next n words

Predict n words

```
In [10]: from inltk.inltk import predict_next_words

         text = """நீராருங் கடலுடுத்த நிலமடந்ைதக் ெகழிலாழுகும்
         சீராரும் வதனெமனத் திகழ்பரதக் கண்டமிதில்
         ெதக்கணமும் அதிற்சிறந்த திராவிடநல் திருநாடும்
         தக்கசிறு பிறநுதலும் தரித்தநறுந் திலகமேம!
         அத்திலக வாசைனேபால் அைனத்துலகும் இன்பமுற
         எத்திசயும் புகழ்மணக்க இருந்தெபருந் தமிழணங்ேக!"""
         predict_next_words(text , 2, 'ta')

         #text --> string in <code-of-language>
         #n --> number of words you want to predict (integer)
```

```
Out[10]: 'நீராருங் கடலுடுத்த நிலமடந்ைதக் ெகழிலாழுகும்\nசீராரும் வதனெமனத் திகழ்பரதக் கண்டமிதில்\nெதக்கணமும் அ
         திற்சிறந்த திராவிடநல் திருநாடும்\nதக்கசிறு பிறநுதலும் தரித்தநறுந் திலகமேம!\nஅத்திலக வாசைனேபால் அைனத்துலகும்
         இன்பமுற\nஎத்திசயும் புகழ்மணக்க இருந்தெபருந் தமிழணங்ேக! தண்டிற்கும்'
```

IV Remove Foreign Languages

Remove Foreign Languages

```
In [12]: from inltk.inltk import remove_foreign_languages

text = """நீராருங் கடலுடுத்த நிலமடந்தைக் ெகழிலாழுகும்
சீராரும் வதனெமனத் திகழ்பரதக் கண்டமிதில்
ெதக்கணமும் அதிற்சிறந்த திராவிடநல் திருநாடும்
தக்கசிறு பிறைநுதலும் தரித்தநறுந் திலகமேம!
அத்திலக வாசைனேபால் அனைத்துலகும் இன்பமுற
எத்திசையும் புகழ்மணக்க இருந்தெபருந் தமிழணங்ேக!"""
remove_foreign_languages(text, 'ta')

#text --> string in one of the supported languages
#<code-of-language> --> code of that language whose words you want to retain
```

```
Out[12]: ['_நீரா',
 'ரு',
 'ங்',
 '_கடல',
 'ு',
 'டு',
 'த்த',
 '_நில',
 'மட',
 'ந்',
 'ை',
 'த',
 'க்',
 '_',
 'ெ',
 'க',
 'ழி',
```

V Get sentence encoding

Get Sentence Encoding

```
In [14]: from inltk.inltk import get_sentence_encoding

get_sentence_encoding(text, 'ta')
```

```
C:\Users\hp\Anaconda3\lib\site-packages\torch\serialization.py:493: SourceChangeWarning: source code of class 'torch.nn.module
s.loss.CrossEntropyLoss' has changed. you can retrieve the original source code by accessing the object's source attribute or s
et `torch.nn.Module.dump_patches = True` and use the patch tool to revert the changes.
  warnings.warn(msg, SourceChangeWarning)
C:\Users\hp\Anaconda3\lib\site-packages\torch\serialization.py:493: SourceChangeWarning: source code of class 'fastai.text.mode
ls.awd_lstm.AWD_LSTM' has changed. you can retrieve the original source code by accessing the object's source attribute or set
`torch.nn.Module.dump_patches = True` and use the patch tool to revert the changes.
  warnings.warn(msg, SourceChangeWarning)
C:\Users\hp\Anaconda3\lib\site-packages\torch\serialization.py:493: SourceChangeWarning: source code of class 'torch.nn.module
s.sparse.Embedding' has changed. you can retrieve the original source code by accessing the object's source attribute or set `t
orch.nn.Module.dump_patches = True` and use the patch tool to revert the changes.
  warnings.warn(msg, SourceChangeWarning)
C:\Users\hp\Anaconda3\lib\site-packages\torch\serialization.py:493: SourceChangeWarning: source code of class 'fastai.text.mode
ls.awd_lstm.EmbeddingDropout' has changed. you can retrieve the original source code by accessing the object's source attribute
or set `torch.nn.Module.dump_patches = True` and use the patch tool to revert the changes.
```

```
  warnings.warn(msg, SourceChangeWarning)
C:\Users\hp\Anaconda3\lib\site-packages\torch\serialization.py:493: SourceChangeWarning: source code of class 'fastai.text.mode
ls.awd_lstm.LinearDecoder' has changed. you can retrieve the original source code by accessing the object's source attribute or
set `torch.nn.Module.dump_patches = True` and use the patch tool to revert the changes.
  warnings.warn(msg, SourceChangeWarning)
C:\Users\hp\Anaconda3\lib\site-packages\torch\serialization.py:493: SourceChangeWarning: source code of class 'torch.nn.module
s.linear.Linear' has changed. you can retrieve the original source code by accessing the object's source attribute or set `torc
h.nn.Module.dump_patches = True` and use the patch tool to revert the changes.
  warnings.warn(msg, SourceChangeWarning)
```

```
Out[14]: array([-0.049043,  0.264813,  0.010338, -0.083325, ...,  0.027954, -0.013759, -0.238054, -0.015276], dtype=float32)
```

Task-2

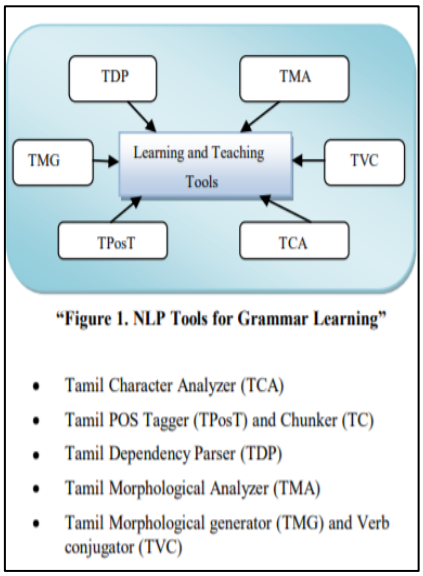
2. Research Papers which are related to regional languages in NLP perspective

- Makwana, Monika T., and Deepak C. Vegda. "Survey: Natural Language Parsing For Indian Languages." *arXiv preprint arXiv:1501.07005* (2015).

In this paper survey is done for different regional languages and the different ways to predict the language and try some applications based on these surveys done by them.

- Dhanalakshmi, V., and S. Rajendran. "Natural language processing tools for tamil grammar learning and teaching." *International journal of Computer Applications* (2010): 0975-8887.

In this paper the idea is that we all know communication is so important and to communicate we should know the language and its grammar rules so they decide to make learn grammar rules of Tamil Language using the NLP technique.



"Figure 1. NLP Tools for Grammar Learning"

- Tamil Character Analyzer (TCA)
- Tamil POS Tagger (TPosT) and Chunker (TC)
- Tamil Dependency Parser (TDP)
- Tamil Morphological Analyzer (TMA)
- Tamil Morphological generator (TMG) and Verb conjugator (TVC)

- Dhanalakshmi, V., R. U. Rekha, Arun Kumar, K. P. Soman, and S. Rajendran. "Morphological analyzer for agglutinative languages using machine learning approaches." In *2009 International Conference on Advances in Recent Technologies in Communication and Computing*, pp. 433-435. IEEE, 2009.

In this paper Morphological analyses for Agglutinative languages is done with the help of some ML techniques and even morphemes also help to do this. Tamil is one of the Agglutinative languages. An agglutinative language is a type of synthetic language with morphology that primarily uses agglutination. Words may contain different morphemes to determine their meanings, but all of these morphemes remain, in every aspect, unchanged after their unions.

Agglutinative languages have generally one grammatical category per affix while fusional languages have multiple.

Examples of Agglutinative languages:

- Indigenous languages of the Americas. Algonquian languages.
- Austronesian languages. Tagalog.
- Niger–Congo languages. Bantu languages.
- Berber languages.
- Dravidian languages. Tamil, Telegu, Malayalam, Kannada
- Eskimo–Aleut languages. Aleut.
- Kartvelian languages.
- Turkic languages. Turkish.

Reference:

1. Makwana, Monika T., and Deepak C. Vegda. "Survey: Natural Language Parsing For Indian Languages." *arXiv preprint arXiv:1501.07005* (2015).

2. Dhanalakshmi, V., and S. Rajendran. "Natural language processing tools for tamil grammar learning and teaching." *International journal of Computer Applications* (2010): 0975-8887.
3. Dhanalakshmi, V., R. U. Rekha, Arun Kumar, K. P. Soman, and S. Rajendran. "Morphological analyzer for agglutinative languages using machine learning approaches." In *2009 International Conference on Advances in Recent Technologies in Communication and Computing*, pp. 433-435. IEEE, 2009.

Task-3

3. Applications for functionalities available in inltk for regional Languages

- Smartphone users in India crossed 500 million in 2019. Businesses are feeling a need to increase user engagement at the local level. NLP can go a long way in achieving that by improving search accuracy (Google Assistant now supports multiple Indian Languages), chatbots and virtual agents, etc
- Text to speech, speech to text
- Digitisation of Indian Manuscripts
- Signboard Translation from Vernacular Languages to make travel more accessible.
- Question Answering Systems
- Automated Learning Systems for Learning/Teaching

Task-4

4. Embedding

Word Embedding using Word2Vec

What is word Embedding?

It is language modelling techniques which helps to map the words to vectors i.e. to real numbers. With the help of this we can represent the words in the vectors with many dimensions. There are different models to do word embedding, but here we used Word2Vec which has models for word modelling.

We need to install some packages,

Word Embedding using Word2Vec

In [1]:
```
#The basic idea of word embedding is words that occur in similar context tend
#For generating word vectors in Python, modules needed are nltk and gensim.
#pip install gensim
```

In [2]:
```
# importing all necessary modules
from nltk.tokenize import sent_tokenize, word_tokenize
import warnings

warnings.filterwarnings(action = 'ignore')

import gensim
from gensim.models import Word2Vec
```

Now taking some sample text,

In [19]:
```
sample = """Alice was beginning to get very tired of sitting by her sister on
So she was considering in her own mind (as well as she could, for the hot day

There was nothing so very remarkable in that; nor did Alice think it so very m
 """
```

We will try to remove the spaces

In [20]:
```
# Replaces escape character with space
f = sample.replace("\n", " ")
f
```

Out[20]:
```
"Alice was beginning to get very tired of sitting by her sister on the bank,
and of having nothing to do: once or twice she had peeped into the book her s
ister was reading, but it had no pictures or conversations in it, `and what i
s the use of a book,' thought Alice `without pictures or conversation?' So sh
e was considering in her own mind (as well as she could, for the hot day made
her feel very sleepy and stupid), whether the pleasure of making a daisy-chai
n would be worth the trouble of getting up and picking the daisies, when sudd
enly a White Rabbit with pink eyes ran close by her.  There was nothing so ve
ry remarkable in that; nor did Alice think it so very much out of the way to
hear the Rabbit say to itself, `Oh dear! Oh dear! I shall be late!' (when she
thought it over afterwards, it occurred to her that she ought to have wondere
d at this, but at the time it all seemed quite natural); but when the Rabbit
actually took a watch out of its waistcoat-pocket, and looked at it, and then
hurried on, Alice started to her feet, for it flashed across her mind that sh
e had never before seen a rabbit with either a waistcoat-pocket, or a watch t
o take out of it, and burning with curiosity, she ran across the field after
it, and fortunately was just in time to see it pop down a large rabbit-hole u
nder the hedge.  "
```

Tokenize the text data,

```
In [21]: #tokenize
         tokens = word_tokenize(f)

In [22]: tokens

Out[22]: ['Alice',
          'was',
          'beginning',
          'to',
          'get',
          'very',
          'tired',
          'of',
          'sitting',
          'by',
          'her',
          'sister',
          'on',
          'the',
          'bank',
          ',',
          'and',
          'of',
          'having',
          'nothing'
```

We will now use the model CBOW (Continuous bag of words),

```
In [33]: print(len(tokens))
         # Create CBOW model (Continuous Bag of Words)
         model1 = gensim.models.Word2Vec(tokens, min_count = 1,
                                         size = 250, window = 5)

         290
```

Display the vectors for each words which are tokenized,

In [40]:
```
print(model1['a'])
```

```
[ 1.76853122e-04 -7.40627816e-04  6.11582291e-05 -6.64107327e-04
  6.84551080e-04  7.64624914e-04  1.79170188e-03 -8.35132902e-04
 -7.29168998e-04  1.60134479e-03 -1.57560827e-03 -5.31373371e-04
  9.85108782e-04  2.60035391e-04 -7.18208204e-04  3.86758387e-04
 -4.09385284e-05  2.04769592e-03  1.73369795e-03 -3.58864272e-05
  1.38477690e-03  4.91368883e-05 -1.97517616e-03  3.62917752e-04
 -7.54107896e-04 -1.24532948e-04 -1.77069614e-03  3.76330339e-04
  1.40519685e-03  6.53162642e-05 -1.76102004e-03  1.54251698e-03
  1.35963690e-03  1.83639361e-03 -7.05297745e-04  1.31731795e-03
  5.64642891e-04  1.15872920e-03 -7.10483000e-04 -1.40704459e-03
  9.64830804e-04  1.70863199e-03  3.43400898e-05  1.79468421e-04
 -1.20703783e-03 -7.75004883e-05  8.86528287e-04 -6.85241015e-04
  1.46281358e-03 -8.25771363e-04  5.58297266e-04  2.72742473e-04
  8.65505601e-04  3.76557582e-04 -2.07669684e-03  1.54004886e-03
 -1.30327034e-03 -1.30896631e-04  5.89768286e-04  3.22790700e-04
  1.67774560e-03  1.96760008e-03 -1.25777163e-03  6.80854500e-05
 -3.24890949e-04 -4.76796267e-04 -3.11195792e-04 -1.53119978e-03
  1.02296495e-03  5.59578475e-04  2.74357280e-05 -8.63678695e-04
 -1.66197878e-03  1.13538292e-03 -9.04287561e-04  1.94128766e-03
  1.69074931e-03 -3.33500619e-04  1.19542715e-03  9.46634740e-04
 -1.61169854e-03 -5.47373318e-04 -1.71430584e-03 -1.81342300e-03
  4.53987624e-04  3.80080746e-04  1.39095972e-03 -1.54118997e-03
 -1.92477833e-03  1.10012756e-04 -1.68688549e-03  1.72271824e-03
 -1.69013394e-03 -6.74453549e-05 -1.16361072e-03 -1.44699603e-04
  1.50835642e-03  2.57608044e-04 -2.60334811e-04 -1.86689151e-03
  8.85363726e-04 -3.68590350e-04 -2.00237380e-03 -1.70579948e-03
 -8.50243960e-04 -4.67175123e-04  6.78728975e-04  1.65897480e-03
 -1.44679321e-03  1.38655188e-03  6.58272707e-04 -8.54098616e-05
 -3.77211341e-04 -1.42180489e-03 -2.95609731e-04 -1.27545919e-03
 -1.47760916e-03  9.08394140e-05 -1.02474634e-03  6.02130487e-04
 -1.35550380e-03 -1.25317776e-03 -1.14038950e-04 -1.94081222e-03
  6.02216569e-05  7.05342100e-04  2.13179295e-03 -1.08400034e-03
```

Task-5

5. Chunking

Chunking for LOR Sample 1

Chunking & Chunking of LOR 1

In [3]:
```
import nltk
doc1 = """I am pleased to recommend MrXX for an MS in Computer Science at your esteemed university. I have known him since his s
I first got to know X in the course of Database Management Systems, CSE-2004. In the first week of the course, I was surprised t
I observed that X had a keen interest and was fully involved in the course when I saw his performances during the Lab sessions, w
As a part of the course, students are required to develop a project, with a fully functional Database System consisting of the c
I was pleased to know that he applied the concepts in his internship extensively to build a professional tool for Intellect Desig
X makes a strong candidate for your Master's program majoring in Computer Science. His proposed candidature has my endorsement w
"""
```

In [4]:
```
token = nltk.word_tokenize(doc1) #tokenize the words
```

In [5]:
```
postags = nltk.pos_tag(token) #do pos tagging of the doc
print(postags)
```

```
[('I', 'PRP'), ('am', 'VBP'), ('pleased', 'JJ'), ('to', 'TO'), ('recommend', 'VB'), ('MrXX', 'NNP'), ('for', 'IN'), ('an', 'D
T'), ('MS', 'NNP'), ('in', 'IN'), ('Computer', 'NNP'), ('Science', 'NNP'), ('at', 'IN'), ('your', 'PRP$'), ('esteemed', 'JJ'),
('university', 'NN'), ('.', '.'), ('I', 'PRP'), ('have', 'VBP'), ('known', 'VBN'), ('him', 'PRP'), ('since', 'IN'), ('his', 'PR
P$'), ('second', 'JJ'), ('year', 'NN'), ('.', '.'), ('He', 'PRP'), ('was', 'VBD'), ('my', 'PRP$'), ('student', 'NN'), ('in', 'I
N'), ('the', 'DT'), ('3rd', 'CD'), ('semester', 'NN'), ('(', '('), ('2nd', 'CD'), ('year', 'NN'), (')', ')'), (',', ','), ('whe
re', 'WRB'), ('I', 'PRP'), ('taught', 'VBD'), ('him', 'PRP'), ('the', 'DT'), ('course', 'NN'), ('of', 'IN'), ('Database', 'NN
P'), ('Management', 'NNP'), ('Systems', 'NNPS'), ('.', '.'), ('I', 'PRP'), ('first', 'RB'), ('got', 'VBD'), ('to', 'TO'), ('kno
w', 'VB'), ('X', 'NNP'), ('in', 'IN'), ('the', 'DT'), ('course', 'NN'), ('of', 'IN'), ('Database', 'NNP'), ('Management', 'NN
P'), ('Systems', 'NNPS'), (',', ','), ('CSE-2004', 'NNP'), ('.', '.'), ('In', 'IN'), ('the', 'DT'), ('first', 'JJ'), ('week',
'NN'), ('of', 'IN'), ('the', 'DT'), ('course', 'NN'), (',', ','), ('I', 'PRP'), ('was', 'VBD'), ('surprised', 'VBN'), ('to', 'T
O'), ('know', 'VB'), ('that', 'DT'), ('X', 'NNP'), (',', ','), ('an', 'DT'), ('Electronics', 'NNS'), ('and', 'CC'), ('Instrumen
tation', 'NNP'), ('student', 'NN'), (',', ','), ('had', 'VBD'), ('taken', 'VBN'), ('up', 'RP'), ('a', 'DT'), ('computer', 'N
N'), ('science', 'NN'), ('core', 'NN'), ('course', 'NN'), ('.', '.'), ('Initially', 'NNP'), ('I', 'PRP'), ('was', 'VBD'), ('dou
btful', 'JJ'), ('about', 'IN'), ('a', 'DT'), ('non-CS', 'JJ'), ('student', 'NN'), ('I', 'NNP'), ('s', 'NN'), ('approach', 'N
```

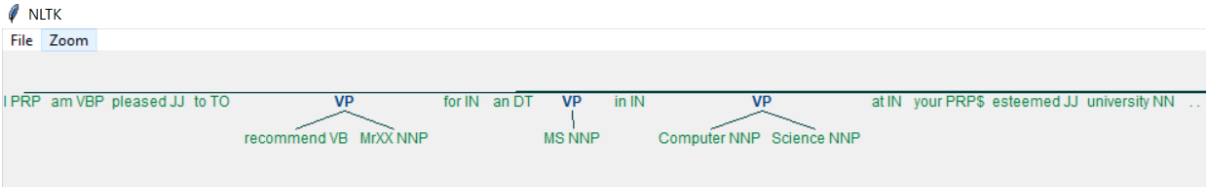
In [46]:
```
grammar1 = "VP: {<VB.?>*<NNP>+<NN>?}" #chunk verb 0 or more times and after that any thing NNP 1 or more times NN should not occ
cp1 = nltk.RegexpParser(grammar1) #RE is ready in Cp
```

In [47]:
```
result = cp1.parse(postags) #entire doc is parsed through the filter
```

In [48]:
```
print(result)
```

```
(S
  I/PRP
  am/VBP
  pleased/JJ
  to/TO
  (VP recommend/VB MrXX/NNP)
  for/IN
  an/DT
  (VP MS/NNP)
  in/IN
  (VP Computer/NNP Science/NNP)
  at/IN
  your/PRP$
  esteemed/JJ
  university/NN
  ./.
  I/PRP
  have/VBP
  known/VBN
  him/PRP
```

In [49]:
```
result.draw()
```



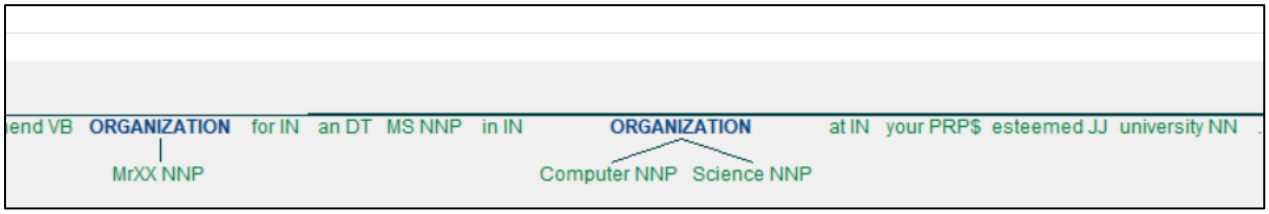
```
In [17]: #NER
         chunks1 = nltk.ne_chunk(postags) #chunking the tagged sentence

In [18]: print(chunks1)
```

```
(S
  I/PRP
  am/VBP
  pleased/JJ
  to/TO
  recommend/VB
  (ORGANIZATION MrXX/NNP)
  for/IN
  an/DT
  MS/NNP
  in/IN
  (ORGANIZATION Computer/NNP Science/NNP)
  at/IN
  your/PRP$
  esteemed/JJ
  university/NN
  ./.
  I/PRP
  have/VBP
  known/VBN
```

```
In [12]: chunks1.draw()
```

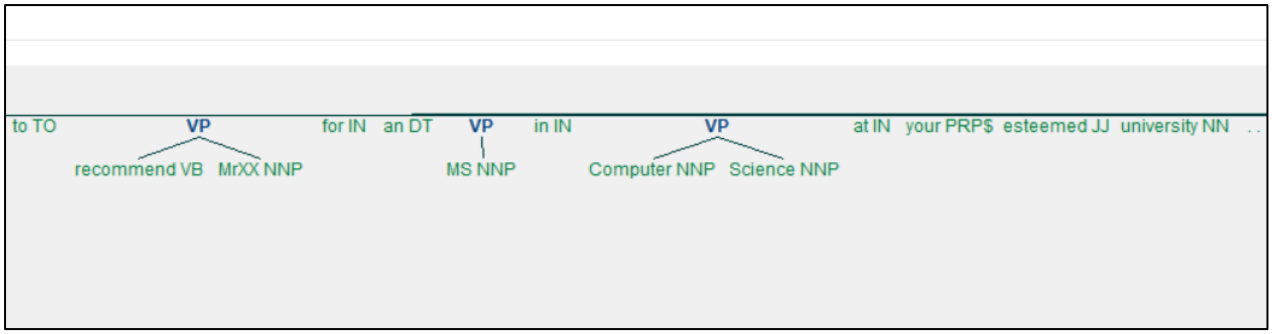
Chunks tree




```
In [44]: #chinking
         #we're removing from the chunk one or more verbs, prepositions, determiners, or the word 'to'.
         grammar = r"""Chunk: {<.*>+}
                                 }<VB.?|IN|DT|TO>+{"""
         cp = nltk.RegexpParser(grammar)
         result = cp.parse(postags)
         print(result)
```

```
(S
  (Chunk I/PRP)
  am/VBP
  (Chunk pleased/JJ)
  to/TO
  recommend/VB
  (Chunk MrXX/NNP)
  for/IN
  an/DT
  (Chunk MS/NNP)
  in/IN
  (Chunk Computer/NNP Science/NNP)
  at/IN
  (Chunk your/PRP$ esteemed/JJ university/NN ./. I/PRP)
  have/VBP
  known/VBN
  (Chunk him/PRP)
  since/IN
  (Chunk his/PRP$ second/JJ year/NN ./. He/PRP)
  was/VBD
```

Chinking Tree



Chunking for LOR Sample-2

Chunking & Chinking of LOR 2

In [20]:
```
doc2 = """I am happy to write this letter of recommendation for AAA who intends to pursue a Master's degree at your prestigious
He has also displayed his dedication and willingness to learn on numerous occasions. On top of this, he is also very efficient i
During the time that I have known him, he had encountered various challenges which he had addressed through his well-directed ef
Apart from his technical skills, his talent for writing and documentation were also quite evident when I had a look at his proje
"""
```

In [21]:
```
tokens = nltk.word_tokenize(doc2) #tokenize the sentences
```

In [23]:
```
pos_tags = nltk.pos_tag(tokens) #do tagging for tokenized sentences
print(pos_tags)
```

```
[('I', 'PRP'), ('am', 'VBP'), ('happy', 'JJ'), ('to', 'TO'), ('write', 'VB'), ('this', 'DT'), ('letter', 'NN'), ('of', 'IN'),
('recommendation', 'NN'), ('for', 'IN'), ('AAA', 'NNP'), ('who', 'WP'), ('intends', 'VBZ'), ('to', 'TO'), ('pursue', 'VB'),
('a', 'DT'), ('Master', 'NN'), ("'s", 'POS'), ('degree', 'NN'), ('at', 'IN'), ('your', 'PRP$'), ('prestigious', 'JJ'), ('univer
sity', 'NN'), ('.', '.'), ('I', 'PRP'), ('have', 'VBP'), ('known', 'VBN'), ('him', 'PRP'), ('only', 'RB'), ('for', 'IN'), ('th
e', 'DT'), ('past', 'JJ'), ('year', 'NN'), ('and', 'CC'), ('it', 'PRP'), ('was', 'VBD'), ('easy', 'JJ'), ('to', 'TO'), ('observ
e', 'VB'), ('that', 'IN'), ('he', 'PRP'), ('is', 'VBZ'), ('a', 'DT'), ('quick', 'JJ'), ('learner', 'NN'), ('and', 'CC'), ('hard
working', 'NN'), ('.', '.'), ('I', 'PRP'), ('handled', 'VBD'), ('the', 'DT'), ('course', 'NN'), ('Natural', 'NNP'), ('Languag
e', 'NNP'), ('Processing', 'NNP'), ('for', 'IN'), ('him', 'PRP'), ('during', 'IN'), ('his', 'PRP$'), ('sixth', 'JJ'), ('semeste
r', 'NN'), ('.', '.'), ('He', 'PRP'), ('has', 'VBZ'), ('also', 'RB'), ('displayed', 'VBN'), ('his', 'PRP$'), ('dedication', 'N
N'), ('and', 'CC'), ('willingness', 'NN'), ('to', 'TO'), ('learn', 'VB'), ('on', 'IN'), ('numerous', 'JJ'), ('occasions', 'NN
S'), ('.', '.'), ('On', 'IN'), ('top', 'NN'), ('of', 'IN'), ('this', 'DT'), (',', ','), ('he', 'PRP'), ('is', 'VBZ'), ('also',
'RB'), ('very', 'RB'), ('efficient', 'JJ'), ('in', 'IN'), ('completing', 'VBG'), ('any', 'DT'), ('task', 'NN'), ('that', 'WD
T'), ('is', 'VBZ'), ('given', 'VBN'), ('to', 'TO'), ('him', 'PRP'), (',', ','), ('never', 'RB'), ('one', 'CD'), ('to', 'TO'),
('complain', 'VB'), ('irrespective', 'JJ'), ('of', 'IN'), ('any', 'DT'), ('circumstances', 'NNS'), ('.', '.'), ('I', 'PRP'),
('remember', 'VBP'), ('him', 'PRP'), ('submitting', 'VBG'), ('NLP', 'NNP'), ('weekly', 'JJ'), ('tasks', 'NNS'), ('well', 'RB'),
('ahead', 'RB'), ('of', 'IN'), ('the', 'DT'), ('specified', 'JJ'), ('time', 'NN'), ('period', 'NN'), ('.', '.'), ('Among', 'I
```

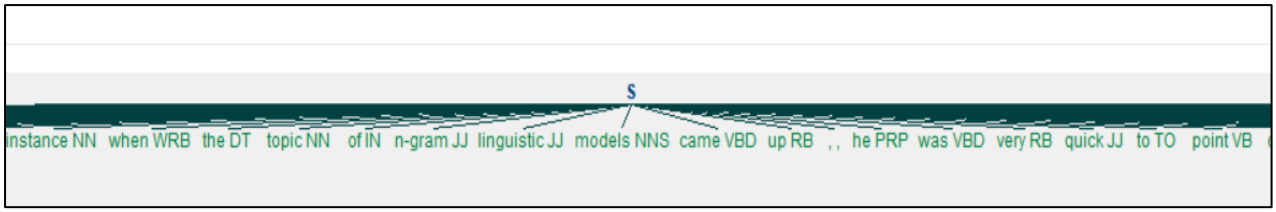
In [24]:
```
grammar2 = "NN: {<DT.?><JJ?><.NN.>}"
cp2 = nltk.RegexpParser(grammar2)
```

In [26]:
```
result2 = cp2.parse(pos_tags)
```

In [27]:
```
print(result2)
```

```
(S
  I/PRP
  am/VBP
  happy/JJ
  to/TO
  write/VB
  this/DT
  letter/NN
  of/IN
  recommendation/NN
  for/IN
  AAA/NNP
  who/WP
  intends/VBZ
  to/TO
  pursue/VB
  a/DT
  Master/NN
  's/POS
  degree/NN
```

In [28]:
```
result2.draw()
```



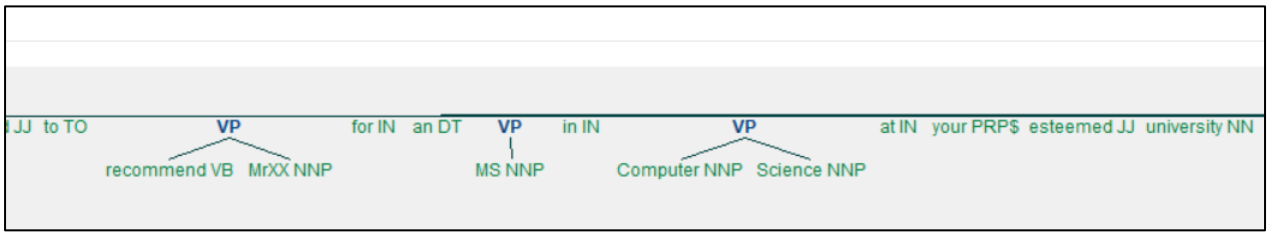

```
In [30]: #NER
         chunks2 = nltk.ne_chunk(pos_tags)

In [32]: print(chunks2)
```

```
(S
  I/PRP
  am/VBP
  happy/JJ
  to/TO
  write/VB
  this/DT
  letter/NN
  of/IN
  recommendation/NN
  for/IN
  AAA/NNP
  who/WP
  intends/VBZ
  to/TO
  pursue/VB
  a/DT
  Master/NN
  's/POS
  degree/NN
```

```
In [33]: chunks2.draw()
```

Chunk Tree

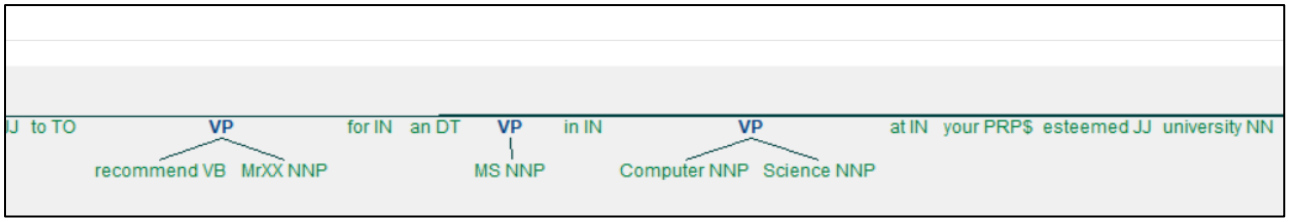



```
In [42]: #chinking
         #we're removing from the chink one or more verbs, prepositions, determiners, or the word 'to'.
         grammar = r"""Chunk: {<.*>+}
                                 }<VB.?|IN|DT|TO>+{"""
         cp = nltk.RegexpParser(grammar)
         result = cp.parse(pos_tags)
         print(result)
```

```
(S
  (Chunk I/PRP)
  am/VBP
  (Chunk happy/JJ)
  to/TO
  write/VB
  this/DT
  (Chunk letter/NN)
  of/IN
  (Chunk recommendation/NN)
  for/IN
  (Chunk AAA/NNP who/WP)
  intends/VBZ
  to/TO
  pursue/VB
  a/DT
  (Chunk Master/NN 's/POS degree/NN)
  at/IN
  (Chunk your/PRP$ prestigious/JJ university/NN ./. I/PRP)
  have/VBP
```

```
In [52]: result.draw()
```

Chinking Trees


JJ to TO
VP
recommend VB
MrXX NNP
for IN
an DT
VP
MS NNP
in IN
VP
Computer NNP
Science NNP
at IN
your PRP$
esteemed JJ
university NN