```
pip install nltk
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.7/dist-packages (3.2.5)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from nltk) (1.15.0)
```

```
[2] import nltk
    nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

Word Tokenize

```
[3] from nltk.tokenize import word_tokenize
    s = '''This is the begining of the nlp practical class.'''
    word_tokenize(s)
```

```
['This',
 'is',
 'the',
 'begining',
 'of',
 'the',
 'nlp',
 'practical',
 'class',
 '.']
```

Sent Tokenize

```
[4] from nltk.tokenize import sent_tokenize, word_tokenize
    sent_tokenize(s)
```

```
['This is the begining of the nlp practical class.']
```

Stemming words with NLTK

```
[5] from nltk.stem import PorterStemmer
    lab = PorterStemmer()
    words=["discover", "discovery", "discovering", "discovered", "discovers","This is the begining of the nlp practical class.","swim", "swims", "swimming", "swimmer", "swimmers"]
    for w in words:
        print(w, " : ", lab.stem(w))
```

```
discover  :  discov
discovery  :  discoveri
discovering  :  discov
discovered  :  discov
discovers  :  discov
This is the begining of the nlp practical class.  :  this is the begining of the nlp practical class.
swim  :  swim
swims  :  swim
swimming  :  swim
swimmer  :  swimmer
swimmers  :  swimmer
```

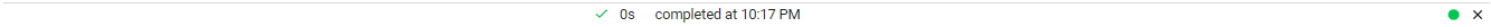
Removing stop words with NLTK

```
[6] import nltk
    nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

```
[7] from nltk.corpus import stopwords
    from nltk.tokenize import word_tokenize

    example_sent = """This is the begining of the nlp practical class."""

    stop_words = set(stopwords.words('english'))

    word_tokens = word_tokenize(example_sent)

    filtered_sentence = [w for w in word_tokens if not w.lower() in stop_words]

    filtered_sentence = []

    for w in word_tokens:
      if w not in stop_words:
        filtered_sentence.append(w)

    print(word_tokens)
    print(filtered_sentence)
```

```
['This', 'is', 'the', 'begining', 'of', 'the', 'nlp', 'practical', 'class', '.']
['This', 'begining', 'nlp', 'practical', 'class', '.']
```


0s
completed at 10:17 PM

N-Grams

```
[1] pip install nltk
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.7/dist-packages (3.2.5)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from nltk) (1.15.0)
```

```
[2] import nltk
    nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

```
[3] s = """
        The Koneru Lakshmaiah Charities was established as a trust
        in the year 1980 with its official address at Museum road, Governorpet, Vijayawada, Andhra Pradesh 520002
        and started KL College of Engineering in the Academic year 1980-81.
    """
```

```
[4] import re

    def generate_ngrams(s, n):
        s = s.lower()

        s = re.sub(r'[^a-zA-Z0-9\s]', ' ', s)
        token = [token for token in s.split(" ") if token != ""]

        ngrams = zip(*[token[i:] for i in range(n)])
        return [" ".join(ngram) for ngram in ngrams]
```

```
[5] generate_ngrams(s, n=5)
```

```
['\n the koneru lakshmaiah charities',
 'the koneru lakshmaiah charities was',
 'koneru lakshmaiah charities was established',
 'lakshmaiah charities was established as',
 'charities was established as a',
 'was established as a trust',
 'established as a trust \n',
 'as a trust \n in',
 'a trust \n in the',
 'trust \n in the year',
 '\n in the year 1980',
 'in the year 1980 with',
 'the year 1980 with its',
 'year 1980 with its official',
 '1980 with its official address',
 'with its official address at',
 'its official address at museum',
 'official address at museum road',
 'address at museum road governorpet',
 'at museum road governorpet vijayawada',
 'museum road governorpet vijayawada andhra',
 'road governorpet vijayawada andhra pradesh',
 'governorpet vijayawada andhra pradesh 520002',
 'vijayawada andhra pradesh 520002 \n',
 'andhra pradesh 520002 \n and',
 'pradesh 520002 \n and started',
 '520002 \n and started kl',
 '\n and started kl college',
 'and started kl college of',
 'started kl college of engineering',
 'kl college of engineering in',
 'college of engineering in the',
 'of engineering in the academic',
 'engineering in the academic year',
 'in the academic year 1980',
 'the academic year 1980 81',
 'academic year 1980 81 \n']
```

```
[6] generate_ngrams(s, n=3)
```

```
['\n the koneru',
 'the koneru lakshmaiah',
 'koneru lakshmaiah charities',
 'lakshmaiah charities was',
 'charities was established',
 'was established as',
 'established as a',
 'as a trust',
 'a trust \n',
 'trust \n in',
 '\n in the',
 'in the year',
 'the year 1980',
 'year 1980 with',
 '1980 with its',
 'with its official',
 'its official address',
 'official address at',
 'address at museum',
 'at museum road',
 'museum road governorpet',
 'road governorpet vijayawada',
 'governorpet vijayawada andhra',
 'vijayawada andhra pradesh',
 'andhra pradesh 520002',
 'pradesh 520002 \n',
 '520002 \n and'
```

```
'\n and started',
'and started kl',
'started kl college',
'kl college of',
'college of engineering',
'of engineering in',
'engineering in the',
'in the academic',
'the academic year',
'academic year 1980',
'year 1980 81',
'1980 81 \n']
```

[7]
```
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

[8]
```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

example_sent = """The Koneru Lakshmaiah Charities was established as a trust
    in the year 1980 with its official address at Museum road, Governorpet, Vijayawada, Andhra Pradesh 520002
    and started KL College of Engineering in the Academic year 1980-81."""

stop_words = set(stopwords.words('english'))

word_tokens = word_tokenize(example_sent)

filtered_sentence = [w for w in word_tokens if not w.lower() in stop_words]

filtered_sentence = []

for w in word_tokens:
  if w not in stop_words:
    filtered_sentence.append(w)

print(word_tokens)
print(filtered_sentence)
```

```
['The', 'Koneru', 'Lakshmaiah', 'Charities', 'was', 'established', 'as', 'a', 'trust', 'in', 'the', 'year', '1980', 'with', 'its', 'official', 'address', 'at', 'Museum', 'road', ',',
['The', 'Koneru', 'Lakshmaiah', 'Charities', 'established', 'trust', 'year', '1980', 'official', 'address', 'Museum', 'road', ',', 'Governorpet', ',', 'Vijayawada', ',', 'Andhra', 'Pr
```

[8]

co-occurrence matrix

```
[ ] import numpy as np
    import pandas as pd
    import nltk
```

```
[ ] from nltk import bigrams
    import itertools
```

```
[ ]
    def generate_co_occurrence_matrix(corpus):
        vocab = set(corpus)
        vocab = list(vocab)
        vocab_index = {word: i for i, word in enumerate(vocab)}

        # Create bigrams from all words in corpus
        bi_grams = list(bigrams(corpus))

        # Frequency distribution of bigrams ((word1, word2), num_occurrences)
        bigram_freq = nltk.FreqDist(bi_grams).most_common(len(bi_grams))

        # Initialise co-occurrence matrix
        # co_occurrence_matrix[current][previous]
        co_occurrence_matrix = np.zeros((len(vocab), len(vocab)))

        # Loop through the bigrams taking the current and previous word,
        # and the number of occurrences of the bigram.
        for bigram in bigram_freq:
            current = bigram[0][1]
            previous = bigram[0][0]
            count = bigram[1]
            pos_current = vocab_index[current]
            pos_previous = vocab_index[previous]
            co_occurrence_matrix[pos_current][pos_previous] = count
        co_occurrence_matrix = np.matrix(co_occurrence_matrix)

        # return the matrix and the index
        return co_occurrence_matrix, vocab_index


    text_data = [['A', 'Jeep', 'is', 'a', 'type', 'of','car'],
                 ['My', 'car','is', 'a', 'Jeep'],
                 ['Jeeps', 'are', 'similar', 'to', 'other', 'cars'],
                 ['Jeep', 'are', 'similar', 'to', 'cars', ',', 'but', 'they', 'are', 'not', 'cars']]

    # Create one list using many lists
    data = list(itertools.chain.from_iterable(text_data))
    matrix, vocab_index = generate_co_occurrence_matrix(data)


    data_matrix = pd.DataFrame(matrix, index=vocab_index,columns=vocab_index)
    print(data_matrix)
```

```
         type    A    ,  Jeeps  cars  but  ...  car   to   of  they   My  are
type      0.0  0.0  0.0    0.0   0.0  0.0  ...  0.0  0.0  0.0   0.0  0.0  0.0
A         0.0  0.0  0.0    0.0   0.0  0.0  ...  0.0  0.0  0.0   0.0  0.0  0.0
,         0.0  0.0  0.0    0.0   1.0  0.0  ...  0.0  0.0  0.0   0.0  0.0  0.0
Jeeps     0.0  0.0  0.0    0.0   0.0  0.0  ...  0.0  0.0  0.0   0.0  0.0  0.0
cars      0.0  0.0  0.0    0.0   0.0  0.0  ...  0.0  1.0  0.0   0.0  0.0  0.0
but       0.0  0.0  1.0    0.0   0.0  0.0  ...  0.0  0.0  0.0   0.0  0.0  0.0
Jeep      0.0  1.0  0.0    0.0   1.0  0.0  ...  0.0  0.0  0.0   0.0  0.0  0.0
other     0.0  0.0  0.0    0.0   0.0  0.0  ...  0.0  1.0  0.0   0.0  0.0  0.0
similar   0.0  0.0  0.0    0.0   0.0  0.0  ...  0.0  0.0  0.0   0.0  0.0  2.0
a         0.0  0.0  0.0    0.0   0.0  0.0  ...  0.0  0.0  0.0   0.0  0.0  0.0
is        0.0  0.0  0.0    0.0   0.0  0.0  ...  1.0  0.0  0.0   0.0  0.0  0.0
not       0.0  0.0  0.0    0.0   0.0  0.0  ...  0.0  0.0  0.0   0.0  0.0  1.0
car       0.0  0.0  0.0    0.0   0.0  0.0  ...  0.0  0.0  1.0   0.0  1.0  0.0
to        0.0  0.0  0.0    0.0   0.0  0.0  ...  0.0  0.0  0.0   0.0  0.0  0.0
of        1.0  0.0  0.0    0.0   0.0  0.0  ...  0.0  0.0  0.0   0.0  0.0  0.0
they      0.0  0.0  0.0    0.0   0.0  1.0  ...  0.0  0.0  0.0   0.0  0.0  0.0
My        0.0  0.0  0.0    0.0   0.0  0.0  ...  1.0  0.0  0.0   0.0  0.0  0.0
are       0.0  0.0  0.0    1.0   0.0  0.0  ...  0.0  0.0  0.0   1.0  0.0  0.0

[18 rows x 18 columns]
```

```
[ ]
```

```
[ ]
```

```
import nltk
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
True
```

```
[ ] from nltk import pos_tag, word_tokenize, RegexpParser
    x=input("enter the sentance: ")
    tagged = pos_tag(word_tokenize(x))
    chunker = RegexpParser("""
                        NP: {<DT>?<JJ>*<NN>}
                        P: {<IN>}
                        V: {<V.*>}
                        PP: {<p> <NP>}
                        VP: {<V> <NP|PP>*}
                        """)
    output = chunker.parse(tagged)
    print("PARTS OF SPEECH:", output)
```

```
enter the sentance: Vaidehi ate the pie
PARTS OF SPEECH: (S Vaidehi/NNP (VP (V ate/VBP) (NP the/DT pie/NN)))
```

Lemmatization

```
[ ] import nltk
    nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
True
```

```
[ ] # import these modules
    from nltk.stem import WordNetLemmatizer

    lemmatizer = WordNetLemmatizer()

    print("rocks :", lemmatizer.lemmatize("rocks"))
    print("corpora :", lemmatizer.lemmatize("corpora"))
    print("connections :", lemmatizer.lemmatize("connections"))
    print("better :", lemmatizer.lemmatize("better", pos ="a"))
```

```
rocks : rock
corpora : corpus
connections : connection
better : good
```

Expand contractions in Text Processing

```
[ ] !pip install contractions
```

```
Collecting contractions
  Downloading contractions-0.1.66-py2.py3-none-any.whl (8.0 kB)
Collecting textsearch>=0.0.21
  Downloading textsearch-0.0.21-py2.py3-none-any.whl (7.5 kB)
Collecting pyahocorasick
  Downloading pyahocorasick-1.4.4-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (106 kB)
     |████████████████████████████████| 106 kB 5.1 MB/s
Collecting anyascii
  Downloading anyascii-0.3.0-py3-none-any.whl (284 kB)
     |████████████████████████████████| 284 kB 53.7 MB/s
Installing collected packages: pyahocorasick, anyascii, textsearch, contractions
Successfully installed anyascii-0.3.0 contractions-0.1.66 pyahocorasick-1.4.4 textsearch-0.0.21
```

```
[ ] import contractions
    # contracted text
    text = '''you're happy now.
              We ain't all the same.
              I'm there at that time.
              we aren't well.'''

    # creating an empty list
    expanded_words = []
    for word in text.split():
      # using contractions.fix to expand the shortened words
      expanded_words.append(contractions.fix(word))

    expanded_text = ' '.join(expanded_words)
    print('Original text: ' + text)
    print('Expanded_text: ' + expanded_text)
```

```
Original text: you're happy now.
          We ain't all the same.
          I'm there at that time.
          we aren't well.
Expanded_text: you are happy now. We are not all the same. I am there at that time. we are not well.
```

```
# Importing libraries
import nltk
import numpy as np
import pandas as pd
import random
from sklearn.model_selection import train_test_split
import pprint, time

#download the treebank corpus from nltk
nltk.download('treebank')

#download the universal tagset from nltk
nltk.download('universal_tagset')

# reading the Treebank tagged sentences
nltk_data = list(nltk.corpus.treebank.tagged_sents(tagset='universal'))

#print the first two sentences along with tags
print(nltk_data[:2])
```

```
[nltk_data] Downloading package treebank to /root/nltk_data...
[nltk_data]   Unzipping corpora/treebank.zip.
[nltk_data] Downloading package universal_tagset to /root/nltk_data...
[nltk_data]   Unzipping taggers/universal_tagset.zip.
[[('Pierre', 'NOUN'), ('Vinken', 'NOUN'), (',', '.'), ('61', 'NUM'), ('years', 'NOUN'), ('old', 'ADJ'), (',', '.'), ('will', 'VERB'), ('join', 'VERB'), ('the', 'DET'), ('board', 'NOUN
```

[ ]
```
#print each word with its respective tag for first two sentences
for sent in nltk_data[:2]:
  for tuple in sent:
    print(tuple)
```

```
('Pierre', 'NOUN')
('Vinken', 'NOUN')
(',', '.')
('61', 'NUM')
('years', 'NOUN')
('old', 'ADJ')
(',', '.')
('will', 'VERB')
('join', 'VERB')
('the', 'DET')
('board', 'NOUN')
('as', 'ADP')
('a', 'DET')
('nonexecutive', 'ADJ')
('director', 'NOUN')
('Nov.', 'NOUN')
('29', 'NUM')
('.', '.')
('Mr.', 'NOUN')
('Vinken', 'NOUN')
('is', 'VERB')
('chairman', 'NOUN')
('of', 'ADP')
('Elsevier', 'NOUN')
('N.V.', 'NOUN')
(',', '.')
('the', 'DET')
('Dutch', 'NOUN')
('publishing', 'VERB')
('group', 'NOUN')
('.', '.')
```

[ ]
```
# split data into training and validation set in the ratio 80:20
train_set,test_set =train_test_split(nltk_data,train_size=0.80,test_size=0.20,random_state = 101)
```

[ ]
```
# create list of train and test tagged words
train_tagged_words = [ tup for sent in train_set for tup in sent ]
test_tagged_words = [ tup for sent in test_set for tup in sent ]
print(len(train_tagged_words))
print(len(test_tagged_words))
```

```
80310
20366
```

[ ]
```
# check some of the tagged words.
train_tagged_words[:5]
```

```
[('Drink', 'NOUN'),
 ('Carrier', 'NOUN'),
 ('Competes', 'VERB'),
 ('With', 'ADP'),
 ('Cartons', 'NOUN')]
```

[ ]
```
#use set datatype to check how many unique tags are present in training data
tags = {tag for word,tag in train_tagged_words}
print(len(tags))
print(tags)

# check total words in vocabulary
vocab = {word for word,tag in train_tagged_words}
```

```
12
{'NUM', 'NOUN', 'CONJ', 'PRT', '.', 'VERB', 'ADP', 'X', 'DET', 'ADV', 'ADJ', 'PRON'}
```

[ ]
```
# compute Emission Probability
def word_given_tag(word, tag, train_bag = train_tagged_words):
```

```
        tag_list = [pair for pair in train_bag if pair[1]==tag]
        count_tag = len(tag_list)#total number of times the passed tag occurred in train_bag
        w_given_tag_list = [pair[0] for pair in tag_list if pair[0]==word]
    #now calculate the total number of times the passed word occurred as the passed tag.
        count_w_given_tag = len(w_given_tag_list)


        return (count_w_given_tag, count_tag)
```

```
[ ] # compute  Transition Probability
    def t2_given_t1(t2, t1, train_bag = train_tagged_words):
        tags = [pair[1] for pair in train_bag]
        count_t1 = len([t for t in tags if t==t1])
        count_t2_t1 = 0
        for index in range(len(tags)-1):
            if tags[index]==t1 and tags[index+1] == t2:
                count_t2_t1 += 1
        return (count_t2_t1, count_t1)
```

```
[ ] # creating t x t transition matrix of tags, t= no of tags
    # Matrix(i, j) represents P(jth tag after the ith tag)

    tags_matrix = np.zeros((len(tags), len(tags)), dtype='float32')
    for i, t1 in enumerate(list(tags)):
        for j, t2 in enumerate(list(tags)):
            tags_matrix[i, j] = t2_given_t1(t2, t1)[0]/t2_given_t1(t2, t1)[1]

    print(tags_matrix)
```

```
[[1.84219927e-01 3.51660132e-01 1.42806144e-02 2.60621198e-02
  1.19243130e-01 2.07068902e-02 3.74866128e-02 2.02427700e-01
  3.57015361e-03 3.57015361e-03 3.53445187e-02 1.42806140e-03]
 [9.14395228e-03 2.62344331e-01 4.24540639e-02 4.39345129e-02
  2.40094051e-01 1.49133503e-01 1.76826611e-01 2.88252197e-02
  1.31063312e-02 1.68945398e-02 1.25838192e-02 4.65906132e-03]
 [4.06147093e-02 3.49066973e-01 5.48847427e-04 4.39077942e-03
  3.51262353e-02 1.50384188e-01 5.59824370e-02 9.33040585e-03
  1.23490669e-01 5.70801310e-02 1.13611415e-01 6.03732169e-02]
 [5.67514673e-02 2.50489235e-01 2.34833662e-03 1.17416831e-03
  4.50097844e-02 4.01174158e-01 1.95694715e-02 1.21330721e-02
  1.01369865e-01 9.39334650e-03 8.29745606e-02 1.76125243e-02]
 [7.82104954e-02 2.18538776e-01 6.00793920e-02 2.78940029e-03
  9.23720598e-02 8.96899477e-02 9.29084867e-02 2.56410260e-02
  1.72191828e-01 5.25694676e-02 4.61323895e-02 6.87694475e-02]
 [2.28360966e-02 1.10589318e-01 5.43278083e-03 3.06629837e-02
  3.48066315e-02 1.67955801e-01 9.23572779e-02 2.15930015e-01
  1.33609578e-01 8.38858187e-02 6.63904250e-02 3.55432779e-02]
 [6.32751212e-02 3.23588967e-01 1.01240189e-03 1.26550242e-03
  3.87243740e-02 8.47886596e-03 1.69577319e-02 3.45482156e-02
  3.20931405e-01 1.45532778e-02 1.07061505e-01 6.96026310e-02]
 [3.07514891e-03 6.16951771e-02 1.03786280e-02 1.85085520e-01
  1.60868734e-01 2.06419379e-01 1.42225638e-01 7.57255405e-02
  5.68902567e-02 2.57543717e-02 1.76821072e-02 5.41995019e-02]
 [2.28546783e-02 6.35906279e-01 4.31220367e-04 2.87480245e-04
  1.73925534e-02 4.02472317e-02 9.91806854e-03 4.51343954e-02
  6.03708485e-03 1.20741697e-02 2.06410810e-01 3.30602261e-03]
 [2.98681147e-02 3.21955010e-02 6.98215654e-03 1.47401085e-02
  1.39255241e-01 3.39022487e-01 1.19472459e-01 2.28859577e-02
  7.13731572e-02 8.14584941e-02 1.30721495e-01 1.20248254e-02]
 [2.17475723e-02 6.96893215e-01 1.68932043e-02 1.14563107e-02
  6.60194159e-02 1.14563107e-02 8.05825219e-02 2.09708735e-02
  5.24271838e-03 5.24271838e-03 6.33009672e-02 1.94174761e-04]
 [6.83371304e-03 2.12756261e-01 5.01138950e-03 1.41230067e-02
  4.19134386e-02 4.84738052e-01 2.23234631e-02 8.83826911e-02
  9.56719834e-03 3.69020514e-02 7.06150308e-02 6.83371304e-03]]
```

```
[ ] # convert the matrix to a df for better readability
    #the table is same as the transition table shown in section 3 of article
    tags_df = pd.DataFrame(tags_matrix, columns = list(tags), index=list(tags))
    display(tags_df)
```

	NUM	NOUN	CONJ	PRT	.	VERB	ADP	X	DET	ADV	ADJ	PRON
NUM	0.184220	0.351660	0.014281	0.026062	0.119243	0.020707	0.037487	0.202428	0.003570	0.003570	0.035345	0.001428
NOUN	0.009144	0.262344	0.042454	0.043935	0.240094	0.149134	0.176827	0.028825	0.013106	0.016895	0.012584	0.004659
CONJ	0.040615	0.349067	0.000549	0.004391	0.035126	0.150384	0.055982	0.009330	0.123491	0.057080	0.113611	0.060373
PRT	0.056751	0.250489	0.002348	0.001174	0.045010	0.401174	0.019569	0.012133	0.101370	0.009393	0.082975	0.017613
.	0.078210	0.218539	0.060079	0.002789	0.092372	0.089690	0.092908	0.025641	0.172192	0.052569	0.046132	0.068769
VERB	0.022836	0.110589	0.005433	0.030663	0.034807	0.167956	0.092357	0.215930	0.133610	0.083886	0.066390	0.035543
ADP	0.063275	0.323589	0.001012	0.001266	0.038724	0.008479	0.016958	0.034548	0.320931	0.014553	0.107062	0.069603
X	0.003075	0.061695	0.010379	0.185086	0.160869	0.206419	0.142226	0.075726	0.056890	0.025754	0.017682	0.054200
DET	0.022855	0.635906	0.000431	0.000287	0.017393	0.040247	0.009918	0.045134	0.006037	0.012074	0.206411	0.003306
ADV	0.029868	0.032196	0.006982	0.014740	0.139255	0.339022	0.119472	0.022886	0.071373	0.081458	0.130721	0.012025
ADJ	0.021748	0.696893	0.016893	0.011456	0.066019	0.011456	0.080583	0.020971	0.005243	0.005243	0.063301	0.000194
PRON	0.006834	0.212756	0.005011	0.014123	0.041913	0.484738	0.022323	0.088383	0.009567	0.036902	0.070615	0.006834

```
[ ] def Viterbi(words, train_bag = train_tagged_words):
        state = []
        T = list(set([pair[1] for pair in train_bag]))

        for key, word in enumerate(words):
            #initialise list of probability column for a given observation
            p = []
            for tag in T:
                if key == 0:
                    transition_p = tags_df.loc['.', tag]
                else:
                    transition_p = tags_df.loc[state[-1], tag]
```

```
            # compute emission and state probabilities
            emission_p = word_given_tag(words[key], tag)[0]/word_given_tag(words[key], tag)[1]
            state_probability = emission_p * transition_p
            p.append(state_probability)

        pmax = max(p)
        # getting state for which probability is maximum
        state_max = T[p.index(pmax)]
        state.append(state_max)
    return list(zip(words, state))
```

[ ]
```
# Let's test our Viterbi algorithm on a few sample sentences of test dataset
random.seed(1234)      #define a random seed to get same sentences when run multiple times

# choose random 10 numbers
rndom = [random.randint(1,len(test_set)) for x in range(10)]

# list of 10 sents on which we test the model
test_run = [test_set[i] for i in rndom]

# list of tagged words
test_run_base = [tup for sent in test_run for tup in sent]

# list of untagged words
test_tagged_words = [tup[0] for sent in test_run for tup in sent]
```

[ ]
```
#Here We will only test 10 sentences to check the accuracy
#as testing the whole training set takes huge amount of time
start = time.time()
tagged_seq = Viterbi(test_tagged_words)
end = time.time()
difference = end-start

print("Time taken in seconds: ", difference)

# accuracy
check = [i for i, j in zip(tagged_seq, test_run_base) if i == j]

accuracy = len(check)/len(tagged_seq)
print('Viterbi Algorithm Accuracy: ',accuracy*100)
```

```
Time taken in seconds:  52.08190035820007
Viterbi Algorithm Accuracy:  93.77990430622009
```

[ ]
```
#Code to test all the test sentences
#(takes alot of time to run s0 we wont run it here)
# tagging the test sentences()
test_tagged_words = [tup for sent in test_set for tup in sent]
test_untagged_words = [tup[0] for sent in test_set for tup in sent]
test_untagged_words

start = time.time()
tagged_seq = Viterbi(test_untagged_words)
end = time.time()
difference = end-start

print("Time taken in seconds: ", difference)

# accuracy
check = [i for i, j in zip(test_tagged_words, test_untagged_words) if i == j]

accuracy = len(check)/len(tagged_seq)
print('Viterbi Algorithm Accuracy: ',accuracy*100)
```

```
Time taken in seconds:  4609.5222725868225
Viterbi Algorithm Accuracy:  0.0
```

[ ]
```
#To improve the performance,we specify a rule base tagger for unknown words
# specify patterns for tagging
patterns = [
    (r'.*ing$', 'VERB'),              # gerund
    (r'.*ed$', 'VERB'),               # past tense
    (r'.*es$', 'VERB'),               # verb
    (r'.*\'s$', 'NOUN'),              # possessive nouns
    (r'.*s$', 'NOUN'),                # plural nouns
    (r'\*T?\*?-[0-9]+$', 'X'),        # X
    (r'^-?[0-9]+(.[0-9]+)?$', 'NUM'), # cardinal numbers
    (r'.*', 'NOUN')                   # nouns
]

# rule based tagger
rule_based_tagger = nltk.RegexpTagger(patterns)
```

[ ]
```
#modified Viterbi to include rule based tagger in it
def Viterbi_rule_based(words, train_bag = train_tagged_words):
    state = []
    T = list(set([pair[1] for pair in train_bag]))

    for key, word in enumerate(words):
        #initialise list of probability column for a given observation
        p = []
        for tag in T:
            if key == 0:
                transition_p = tags_df.loc['.', tag]
            else:
                transition_p = tags_df.loc[state[-1], tag]

            # compute emission and state probabilities
            emission_p = word_given_tag(words[key], tag)[0]/word_given_tag(words[key], tag)[1]
```

```
            state_probability = emission_p * transition_p
            p.append(state_probability)

        pmax = max(p)
        state_max = rule_based_tagger.tag([word])[0][1]


        if(pmax==0):
            state_max = rule_based_tagger.tag([word])[0][1] # assign based on rule based tagger
        else:
            if state_max != 'X':
                # getting state for which probability is maximum
                state_max = T[p.index(pmax)]


        state.append(state_max)
    return list(zip(words, state))
```

```
[ ] #test accuracy on subset of test data
    start = time.time()
    tagged_seq = Viterbi_rule_based(test_tagged_words)
    end = time.time()
    difference = end-start

    print("Time taken in seconds: ", difference)

    # accuracy
    check = [i for i, j in zip(tagged_seq, test_run_base) if i == j]

    accuracy = len(check)/len(tagged_seq)
    print('Viterbi Algorithm Accuracy: ',accuracy*100)
```

```
# Importing libraries
import nltk
import numpy as np
import pandas as pd
import random
from sklearn.model_selection import train_test_split
import pprint, time

#download the treebank corpus from nltk
nltk.download('treebank')

#download the universal tagset from nltk
nltk.download('universal_tagset')

# reading the Treebank tagged sentences
nltk_data = list(nltk.corpus.treebank.tagged_sents(tagset='universal'))

#print the first two sentences along with tags
print(nltk_data[:2])
```

```
[nltk_data] Downloading package treebank to /root/nltk_data...
[nltk_data]   Unzipping corpora/treebank.zip.
[nltk_data] Downloading package universal_tagset to /root/nltk_data...
[nltk_data]   Unzipping taggers/universal_tagset.zip.
[[('Pierre', 'NOUN'), ('Vinken', 'NOUN'), (',', '.'), ('61', 'NUM'), ('years', 'NOUN'), ('old', 'ADJ'), (',', '.'), ('will', 'VERB'), ('join', 'VERB'), ('the', 'DET'), ('board', 'NOUN
```

[ ]
```
#print each word with its respective tag for first two sentences
for sent in nltk_data[:2]:
  for tuple in sent:
    print(tuple)
```

```
('Pierre', 'NOUN')
('Vinken', 'NOUN')
(',', '.')
('61', 'NUM')
('years', 'NOUN')
('old', 'ADJ')
(',', '.')
('will', 'VERB')
('join', 'VERB')
('the', 'DET')
('board', 'NOUN')
('as', 'ADP')
('a', 'DET')
('nonexecutive', 'ADJ')
('director', 'NOUN')
('Nov.', 'NOUN')
('29', 'NUM')
('.', '.')
('Mr.', 'NOUN')
('Vinken', 'NOUN')
('is', 'VERB')
('chairman', 'NOUN')
('of', 'ADP')
('Elsevier', 'NOUN')
('N.V.', 'NOUN')
(',', '.')
('the', 'DET')
('Dutch', 'NOUN')
('publishing', 'VERB')
('group', 'NOUN')
('.', '.')
```

[ ]
```
# split data into training and validation set in the ratio 80:20
train_set,test_set =train_test_split(nltk_data,train_size=0.80,test_size=0.20,random_state = 101)
```

[ ]
```
# create list of train and test tagged words
train_tagged_words = [ tup for sent in train_set for tup in sent ]
test_tagged_words = [ tup for sent in test_set for tup in sent ]
print(len(train_tagged_words))
print(len(test_tagged_words))
```

```
80310
20366
```

[ ]
```
# check some of the tagged words.
train_tagged_words[:5]
```

```
[('Drink', 'NOUN'),
 ('Carrier', 'NOUN'),
 ('Competes', 'VERB'),
 ('With', 'ADP'),
 ('Cartons', 'NOUN')]
```

[ ]
```
#use set datatype to check how many unique tags are present in training data
tags = {tag for word,tag in train_tagged_words}
print(len(tags))
print(tags)

# check total words in vocabulary
vocab = {word for word,tag in train_tagged_words}
```

```
12
{'NUM', 'NOUN', 'CONJ', 'PRT', '.', 'VERB', 'ADP', 'X', 'DET', 'ADV', 'ADJ', 'PRON'}
```

[ ]
```
# compute Emission Probability
def word_given_tag(word, tag, train_bag = train_tagged_words):
```

```
        tag_list = [pair for pair in train_bag if pair[1]==tag]
        count_tag = len(tag_list)#total number of times the passed tag occurred in train_bag
        w_given_tag_list = [pair[0] for pair in tag_list if pair[0]==word]
    #now calculate the total number of times the passed word occurred as the passed tag.
        count_w_given_tag = len(w_given_tag_list)


        return (count_w_given_tag, count_tag)
```

```
[ ] # compute  Transition Probability
    def t2_given_t1(t2, t1, train_bag = train_tagged_words):
        tags = [pair[1] for pair in train_bag]
        count_t1 = len([t for t in tags if t==t1])
        count_t2_t1 = 0
        for index in range(len(tags)-1):
            if tags[index]==t1 and tags[index+1] == t2:
                count_t2_t1 += 1
        return (count_t2_t1, count_t1)
```

```
[ ] # creating t x t transition matrix of tags, t= no of tags
    # Matrix(i, j) represents P(jth tag after the ith tag)

    tags_matrix = np.zeros((len(tags), len(tags)), dtype='float32')
    for i, t1 in enumerate(list(tags)):
        for j, t2 in enumerate(list(tags)):
            tags_matrix[i, j] = t2_given_t1(t2, t1)[0]/t2_given_t1(t2, t1)[1]

    print(tags_matrix)
```

```
[[1.84219927e-01 3.51660132e-01 1.42806144e-02 2.60621198e-02
  1.19243130e-01 2.07068902e-02 3.74866128e-02 2.02427700e-01
  3.57015361e-03 3.57015361e-03 3.53445187e-02 1.42806140e-03]
 [9.14395228e-03 2.62344331e-01 4.24540639e-02 4.39345129e-02
  2.40094051e-01 1.49133503e-01 1.76826611e-01 2.88252197e-02
  1.31063312e-02 1.68945398e-02 1.25838192e-02 4.65906132e-03]
 [4.06147093e-02 3.49066973e-01 5.48847427e-04 4.39077942e-03
  3.51262353e-02 1.50384188e-01 5.59824370e-02 9.33040585e-03
  1.23490669e-01 5.70801310e-02 1.13611415e-01 6.03732169e-02]
 [5.67514673e-02 2.50489235e-01 2.34833662e-03 1.17416831e-03
  4.50097844e-02 4.01174158e-01 1.95694715e-02 1.21330721e-02
  1.01369865e-01 9.39334650e-03 8.29745606e-02 1.76125243e-02]
 [7.82104954e-02 2.18538776e-01 6.00793920e-02 2.78940029e-03
  9.23720598e-02 8.96899477e-02 9.29084867e-02 2.56410260e-02
  1.72191828e-01 5.25694676e-02 4.61323895e-02 6.87694475e-02]
 [2.28360966e-02 1.10589318e-01 5.43278083e-03 3.06629837e-02
  3.48066315e-02 1.67955801e-01 9.23572779e-02 2.15930015e-01
  1.33609578e-01 8.38858187e-02 6.63904250e-02 3.55432779e-02]
 [6.32751212e-02 3.23588967e-01 1.01240189e-03 1.26550242e-03
  3.87243740e-02 8.47886596e-03 1.69577319e-02 3.45482156e-02
  3.20931405e-01 1.45532778e-02 1.07061505e-01 6.96026310e-02]
 [3.07514891e-03 6.16951771e-02 1.03786280e-02 1.85085520e-01
  1.60868734e-01 2.06419379e-01 1.42225638e-01 7.57255405e-02
  5.68902567e-02 2.57543717e-02 1.76821072e-02 5.41995019e-02]
 [2.28546783e-02 6.35906279e-01 4.31220367e-04 2.87480245e-04
  1.73925534e-02 4.02472317e-02 9.91806854e-03 4.51343954e-02
  6.03708485e-03 1.20741697e-02 2.06410810e-01 3.30602261e-03]
 [2.98681147e-02 3.21955010e-02 6.98215654e-03 1.47401085e-02
  1.39255241e-01 3.39022487e-01 1.19472459e-01 2.28859577e-02
  7.13731572e-02 8.14584941e-02 1.30721495e-01 1.20248254e-02]
 [2.17475723e-02 6.96893215e-01 1.68932043e-02 1.14563107e-02
  6.60194159e-02 1.14563107e-02 8.05825219e-02 2.09708735e-02
  5.24271838e-03 5.24271838e-03 6.33009672e-02 1.94174761e-04]
 [6.83371304e-03 2.12756261e-01 5.01138950e-03 1.41230067e-02
  4.19134386e-02 4.84738052e-01 2.23234631e-02 8.83826911e-02
  9.56719834e-03 3.69020514e-02 7.06150308e-02 6.83371304e-03]]
```

```
[ ] # convert the matrix to a df for better readability
    #the table is same as the transition table shown in section 3 of article
    tags_df = pd.DataFrame(tags_matrix, columns = list(tags), index=list(tags))
    display(tags_df)
```

	NUM	NOUN	CONJ	PRT	.	VERB	ADP	X	DET	ADV	ADJ	PRON
NUM	0.184220	0.351660	0.014281	0.026062	0.119243	0.020707	0.037487	0.202428	0.003570	0.003570	0.035345	0.001428
NOUN	0.009144	0.262344	0.042454	0.043935	0.240094	0.149134	0.176827	0.028825	0.013106	0.016895	0.012584	0.004659
CONJ	0.040615	0.349067	0.000549	0.004391	0.035126	0.150384	0.055982	0.009330	0.123491	0.057080	0.113611	0.060373
PRT	0.056751	0.250489	0.002348	0.001174	0.045010	0.401174	0.019569	0.012133	0.101370	0.009393	0.082975	0.017613
.	0.078210	0.218539	0.060079	0.002789	0.092372	0.089690	0.092908	0.025641	0.172192	0.052569	0.046132	0.068769
VERB	0.022836	0.110589	0.005433	0.030663	0.034807	0.167956	0.092357	0.215930	0.133610	0.083886	0.066390	0.035543
ADP	0.063275	0.323589	0.001012	0.001266	0.038724	0.008479	0.016958	0.034548	0.320931	0.014553	0.107062	0.069603
X	0.003075	0.061695	0.010379	0.185086	0.160869	0.206419	0.142226	0.075726	0.056890	0.025754	0.017682	0.054200
DET	0.022855	0.635906	0.000431	0.000287	0.017393	0.040247	0.009918	0.045134	0.006037	0.012074	0.206411	0.003306
ADV	0.029868	0.032196	0.006982	0.014740	0.139255	0.339022	0.119472	0.022886	0.071373	0.081458	0.130721	0.012025
ADJ	0.021748	0.696893	0.016893	0.011456	0.066019	0.011456	0.080583	0.020971	0.005243	0.005243	0.063301	0.000194
PRON	0.006834	0.212756	0.005011	0.014123	0.041913	0.484738	0.022323	0.088383	0.009567	0.036902	0.070615	0.006834

```
[ ] def Viterbi(words, train_bag = train_tagged_words):
        state = []
        T = list(set([pair[1] for pair in train_bag]))

        for key, word in enumerate(words):
            #initialise list of probability column for a given observation
            p = []
            for tag in T:
                if key == 0:
                    transition_p = tags_df.loc['.', tag]
                else:
                    transition_p = tags_df.loc[state[-1], tag]
```

```
            # compute emission and state probabilities
            emission_p = word_given_tag(words[key], tag)[0]/word_given_tag(words[key], tag)[1]
            state_probability = emission_p * transition_p
            p.append(state_probability)

        pmax = max(p)
        # getting state for which probability is maximum
        state_max = T[p.index(pmax)]
        state.append(state_max)
    return list(zip(words, state))
```

[ ]
```
# Let's test our Viterbi algorithm on a few sample sentences of test dataset
random.seed(1234)      #define a random seed to get same sentences when run multiple times

# choose random 10 numbers
rndom = [random.randint(1,len(test_set)) for x in range(10)]

# list of 10 sents on which we test the model
test_run = [test_set[i] for i in rndom]

# list of tagged words
test_run_base = [tup for sent in test_run for tup in sent]

# list of untagged words
test_tagged_words = [tup[0] for sent in test_run for tup in sent]
```

[ ]
```
#Here We will only test 10 sentences to check the accuracy
#as testing the whole training set takes huge amount of time
start = time.time()
tagged_seq = Viterbi(test_tagged_words)
end = time.time()
difference = end-start

print("Time taken in seconds: ", difference)

# accuracy
check = [i for i, j in zip(tagged_seq, test_run_base) if i == j]

accuracy = len(check)/len(tagged_seq)
print('Viterbi Algorithm Accuracy: ',accuracy*100)
```

```
Time taken in seconds:  52.08190035820007
Viterbi Algorithm Accuracy:  93.77990430622009
```

[ ]
```
#Code to test all the test sentences
#(takes alot of time to run s0 we wont run it here)
# tagging the test sentences()
test_tagged_words = [tup for sent in test_set for tup in sent]
test_untagged_words = [tup[0] for sent in test_set for tup in sent]
test_untagged_words

start = time.time()
tagged_seq = Viterbi(test_untagged_words)
end = time.time()
difference = end-start

print("Time taken in seconds: ", difference)

# accuracy
check = [i for i, j in zip(test_tagged_words, test_untagged_words) if i == j]

accuracy = len(check)/len(tagged_seq)
print('Viterbi Algorithm Accuracy: ',accuracy*100)
```

```
Time taken in seconds:  4609.5222725868225
Viterbi Algorithm Accuracy:  0.0
```

[ ]
```
#To improve the performance,we specify a rule base tagger for unknown words
# specify patterns for tagging
patterns = [
    (r'.*ing$', 'VERB'),              # gerund
    (r'.*ed$', 'VERB'),               # past tense
    (r'.*es$', 'VERB'),               # verb
    (r'.*\'s$', 'NOUN'),              # possessive nouns
    (r'.*s$', 'NOUN'),                # plural nouns
    (r'\*T?\*?-[0-9]+$', 'X'),        # X
    (r'^-?[0-9]+(.[0-9]+)?$', 'NUM'), # cardinal numbers
    (r'.*', 'NOUN')                   # nouns
]

# rule based tagger
rule_based_tagger = nltk.RegexpTagger(patterns)
```

[ ]
```
#modified Viterbi to include rule based tagger in it
def Viterbi_rule_based(words, train_bag = train_tagged_words):
    state = []
    T = list(set([pair[1] for pair in train_bag]))

    for key, word in enumerate(words):
        #initialise list of probability column for a given observation
        p = []
        for tag in T:
            if key == 0:
                transition_p = tags_df.loc['.', tag]
            else:
                transition_p = tags_df.loc[state[-1], tag]

            # compute emission and state probabilities
            emission_p = word_given_tag(words[key], tag)[0]/word_given_tag(words[key], tag)[1]
```

```
            state_probability = emission_p * transition_p
            p.append(state_probability)

        pmax = max(p)
        state_max = rule_based_tagger.tag([word])[0][1]


        if(pmax==0):
            state_max = rule_based_tagger.tag([word])[0][1] # assign based on rule based tagger
        else:
            if state_max != 'X':
                # getting state for which probability is maximum
                state_max = T[p.index(pmax)]


        state.append(state_max)
    return list(zip(words, state))
```

```
[ ] #test accuracy on subset of test data
    start = time.time()
    tagged_seq = Viterbi_rule_based(test_tagged_words)
    end = time.time()
    difference = end-start

    print("Time taken in seconds: ", difference)

    # accuracy
    check = [i for i, j in zip(tagged_seq, test_run_base) if i == j]

    accuracy = len(check)/len(tagged_seq)
    print('Viterbi Algorithm Accuracy: ',accuracy*100)
```

```
import nltk
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
import codecs
from nltk.tokenize import PunktSentenceTokenizer
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer, PorterStemmer
from nltk.corpus import wordnet
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

```
# Remove Stop Words . Word Stemming . Return new tokenised list.


def filteredSentence(sentence):

    filtered_sent = []
    lemmatizer = WordNetLemmatizer()  # lemmatizes the words
    ps = PorterStemmer()  # stemmer stems the root of the word.

    stop_words = set(stopwords.words("english"))
    words = word_tokenize(sentence)

    for w in words:
        if w not in stop_words:
            filtered_sent.append(lemmatizer.lemmatize(ps.stem(w)))
            for i in synonymsCreator(w):
                filtered_sent.append(i)
    return filtered_sent
```

```
# Add synonyms to match list


def synonymsCreator(word):
    synonyms = []

    for syn in wordnet.synsets(word):
        for i in syn.lemmas():
            synonyms.append(i.name())

    return synonyms
```

```
# Cehck and return similarity


def simlilarityCheck(word1, word2):

    word1 = word1 + ".n.01"
    word2 = word2 + ".n.01"
    try:
        w1 = wordnet.synset(word1)
        w2 = wordnet.synset(word2)

        return w1.wup_similarity(w2)

    except:
        return 0
```

```
def simpleFilter(sentence):

    filtered_sent = []
    lemmatizer = WordNetLemmatizer()
    stop_words = set(stopwords.words("english"))
    words = word_tokenize(sentence)

    for w in words:
        if w not in stop_words:
            filtered_sent.append(lemmatizer.lemmatize(w))
            # for i in synonymsCreator(w):
            #   filtered_sent.append(i)
    return filtered_sent


if __name__ == '__main__':

    cricfile = codecs.open("cricketbat.txt", 'r', "utf-8")
    sent2 = cricfile.read().lower()
    vampirefile = codecs.open("vampirebat.txt", 'r', 'utf-8')
    sent1 = vampirefile.read().lower()
    sent3 = "start"

    # FOR TEST , replace the above variables with below sent1 and sent 2
    # sent1 = "the commercial banks are used for finance. all the financial matters are managed by financial banks and they have lots of money, user accounts like salary account and
    # sent2 = "the river bank has water in it and it has fishes trees . lots of water is stored in the banks. boats float in it and animals come and drink water from it."
    # sent3 = "from which bank should i withdraw money"

    while(sent3 != "end"):

        sent3 = input("Enter Query: ").lower()

        filtered_sent1 = []
        filtered_sent2 = []
        filtered_sent3 = []

        counter1 = 0
        counter2 = 0
        sent31_similarity = 0
        sent32_similarity = 0

        filtered_sent1 = simpleFilter(sent1)
        filtered_sent2 = simpleFilter(sent2)
        filtered_sent3 = simpleFilter(sent3)

        for i in filtered_sent3:

            for j in filtered_sent1:
                counter1 = counter1 + 1
                sent31_similarity = sent31_similarity + simlilarityCheck(i, j)

            for j in filtered_sent2:
                counter2 = counter2 + 1
                sent32_similarity = sent32_similarity + simlilarityCheck(i, j)

        filtered_sent1 = []
        filtered_sent2 = []
        filtered_sent3 = []

        filtered_sent1 = filteredSentence(sent1)
        filtered_sent2 = filteredSentence(sent2)
        filtered_sent3 = filteredSentence(sent3)

        sent1_count = 0
        sent2_count = 0

        for i in filtered_sent3:

            for j in filtered_sent1:

                if(i == j):
                    sent1_count = sent1_count + 1
```

```
            for j in filtered_sent2:
                if(i == j):
                    sent2_count = sent2_count + 1

        if((sent1_count + sent31_similarity) > (sent2_count+sent32_similarity)):
            print ("Mammal Bat")
        else:
            print ("Cricket Bat")

        # ------------------------------------------------
        # Sentence1: the river bank has water in it and it has fishes trees . lots of water is stored in the banks. boats float in it and animals come and drink water from it.
        # sentence2: the commercial banks are used for finance. all the financial matters are managed by financial banks and they have lots of money, user accounts like salary accoun
        # query: from which bank should i withdraw money.

        # sen1: any of various nocturnal flying mammals of the order Chiroptera, having membranous wings that extend from the forelimbs to the hind limbs or tail and anatomical adapt
        # sen 2: a cricket wooden bat is used for playing criket. it is rectangular in shape and has handle and is made of wood or plastic and is used by cricket players.
    print ("\nTERMINATED")
```

```
Enter Query: which bat has handle ?
Cricket Bat
Enter Query: quality of bat
Mammal Bat
Enter Query: birth
Mammal Bat
Enter Query: bat type
Mammal Bat
Enter Query: see
Mammal Bat
Enter Query: use
Cricket Bat
Enter Query: end
Mammal Bat

TERMINATED
```