**Assignment 01**

**Assignment Name**: Write a program to implement sentence segmentation and word tokenization

**Name: Nishikant Pawar Class: MSc Cs Part 1**

**Roll No: Date:**

**---------------------------------------------------------------------------------------------------------------------**

**Source Code:**

import nltk

from nltk.tokenize import sent\_tokenize, word\_tokenize

A="My name is Mohib, i am from msccs part1 .I love python .so what about you?"

print("SENTENCE TOKENIZATION")

print(sent\_tokenize(A))

for i in sent\_tokenize(A):

print(i)

print("\n WORD TOKENIZATION")

print(word\_tokenize (A))

for i in word\_tokenize(A):

print(i)

**Output:**

SENTENCE TOKENIZATION

['My name is Mohib, i am from msccs part1 .I love python .so what about you?']

My name is Mohib, i am from msccs part1 .I love python .so what about you?

WORD TOKENIZATION

['My', 'name', 'is', 'Mohib', ',', 'i', 'am', 'from', 'msccs', 'part1', '.I', 'love', 'python', '.so', 'what', 'about', 'you', '?']

My

name

is

Mohib

,

i

am

from

msccs

part1

.I

love

python

.so

what

about

you

?

**Assignment Name 02**

**Assignment Name: Write a program to implement stemming and lemmatization.**

**Name: Nishikant Pawar. Class: MSc CS Part 1**

**Roll No: Date:**

**---------------------------------------------------------------------------------------------------------------------**

**Source Code:**

#implement stemming and lemmization

import nltk

from nltk.stem.porter import PorterStemmer

from nltk.stem import WordNetLemmatizer

porter\_stemmer = PorterStemmer()

word\_data = "Natural language processing (NLP) is the ability of a computer program to understand human language as it is spoken and written"

# First Word tokenization

nltk\_tokens = nltk.word\_tokenize(word\_data)

print("Stemming")

#Next find the roots of the word

for w in nltk\_tokens:

print ("Actual: %s Stem: %s" % (w,porter\_stemmer.stem(w)))

wordnet\_lemmatizer = WordNetLemmatizer()

nltk\_tokens = nltk.word\_tokenize(word\_data)

print("Lemmatization")

for w in nltk\_tokens:

print ("Actual: %s Lemma: %s" % (w,wordnet\_lemmatizer.lemmatize(w)))

**Output:-**

l.py

Stemming

Actual: Natural Stem: natur

Actual: language Stem: languag

Actual: processing Stem: process

Actual: ( Stem: (

Actual: NLP Stem: nlp

Actual: ) Stem: )

Actual: is Stem: is

Actual: the Stem: the

Actual: ability Stem: abil

Actual: of Stem: of

Actual: a Stem: a

Actual: computer Stem: comput

Actual: program Stem: program

Actual: to Stem: to

Actual: understand Stem: understand

Actual: human Stem: human

Actual: language Stem: languag

Actual: as Stem: as

Actual: it Stem: it

Actual: is Stem: is

Actual: spoken Stem: spoken

Actual: and Stem: and

Actual: written Stem: written

Lemmatization

Actual: Natural Lemma: Natural

Actual: language Lemma: language

Actual: processing Lemma: processing

Actual: ( Lemma: (

Actual: NLP Lemma: NLP

Actual: ) Lemma: )

Actual: is Lemma: is

Actual: the Lemma: the

Actual: ability Lemma: ability

Actual: of Lemma: of

Actual: a Lemma: a

Actual: computer Lemma: computer

Actual: program Lemma: program

Actual: to Lemma: to

Actual: understand Lemma: understand

Actual: human Lemma: human

Actual: language Lemma: language

Actual: as Lemma: a

Actual: it Lemma: it

Actual: is Lemma: is

Actual: spoken Lemma: spoken

Actual: and Lemma: and

Actual: written Lemma: written

>>>

**Assignment 03**

**Write a program to implement a tri-gram model.**

**Name: Nishikant Pawar. Roll No: Class: MSc CS Part 1 Date:**

**---------------------------------------------------------------------------------------------------------------------**

**Source Code:**

from nltk.util import\*

n=1

sentence="The Natural Language Toolkit , more commonly NLTK, is a suite of libraries and program for symbolic and statistical natural language processing for English written in the python programming language."

unigram=ngrams(sentence.split(),n)

for item in unigram:

print(item)

**Output:**

('The',)

('Natural',)

('Language',)

('Toolkit,',)

('more',)

('commonly',)

('NLTK,',)

('is',)

('a'

**Assignment No 04**

**Assignment Name : Write a program to implement POS Tagging using HMM and Neural Models.**

**Name: Nishikant Pawar Class: MSc Cs Part 1**

**Roll No: Date:**

**---------------------------------------------------------------------------------------------------------------------**

**Source code:**

import nltk

from nltk import tokenize

nltk.download('punkt')

from nltk import tag

from nltk import chunk

nltk.download('averaged\_perceptron\_tagger')

nltk.download('maxent\_ne\_chunker')

nltk.download('words')

para = "Hello! My name is Beena Kapadia. Today you'll be learning NLTK." sents = tokenize.sent\_tokenize(para)

print("\nsentence tokenization\n===================\n",sents)

# word tokenization

print("\nword tokenization\n===================\n")

for index in range(len(sents)):

 words = tokenize.word\_tokenize(sents[index])

 print(words)

  # POS Tagging

tagged\_words = []

for index in range(len(sents)):

 tagged\_words.append(tag.pos\_tag(words))

print("\nPOS Tagging\n===========\n",tagged\_words)

# chunking

tree = []

for index in range(len(sents)):

 tree.append(chunk.ne\_chunk(tagged\_words[index]))

print("\nchunking\n========\n")

print(tree)

**Output:**

sentence tokenization

===================

['Hello!', 'My name is Beena Kapadia.', "Today you'll be learning NLTK."]

word tokenization

===================

['Hello', '!']

['My', 'name', 'is', 'Beena', 'Kapadia', '.']

['Today', 'you', "'ll", 'be', 'learning', 'NLTK', '.']

POS Tagging

===========

[[('Today', 'NN'), ('you', 'PRP'), ("'ll", 'MD'), ('be', 'VB'), ('learning', 'VBG'), ('NLTK',  'NNP'), ('.', '.')], [('Today', 'NN'), ('you', 'PRP'), ("'ll", 'MD'), ('be', 'VB'), ('learning',  'VBG'), ('NLTK', 'NNP'), ('.', '.')], [('Today', 'NN'), ('you', 'PRP'), ("'ll", 'MD'), ('be',  'VB'), ('learning', 'VBG'), ('NLTK', 'NNP'), ('.', '.')]]

chunking

========

[Tree('S', [('Today', 'NN'), ('you', 'PRP'), ("'ll", 'MD'), ('be', 'VB'), ('learning', 'VBG'),  Tree('ORGANIZATION', [('NLTK', 'NNP')]), ('.', '.')]), Tree('S', [('Today', 'NN'), ('you',  'PRP'), ("'ll", 'MD'), ('be', 'VB'), ('learning', 'VBG'), Tree('ORGANIZATION', [('NLTK',  'NNP')]), ('.', '.')]), Tree('S', [('Today', 'NN'), ('you', 'PRP'), ("'ll", 'MD'), ('be', 'VB'),  ('learning', 'VBG'), Tree('ORGANIZATION', [('NLTK', 'NNP')]), ('.', '.')])]

**Assignment no 05**

**Assignment Name:-**.Write a program to implement syntactic parsing of the given text.

**Name:** nishikant pawar **Roll No:-**

**Class:-** Msc(cs)part1 **Date:-**

**Source code:**

import nltk

from nltk import tokenize

grammar1 = nltk.CFG.fromstring("""

S -> VP

 VP -> VP NP

 NP -> Det NP

 Det -> 'that'

 NP -> singular Noun

 NP -> 'flight'

 VP -> 'Book'

""")

sentence = "Book that flight"

for index in range(len(sentence)):

 all\_tokens = tokenize.word\_tokenize(sentence)

print(all\_tokens)

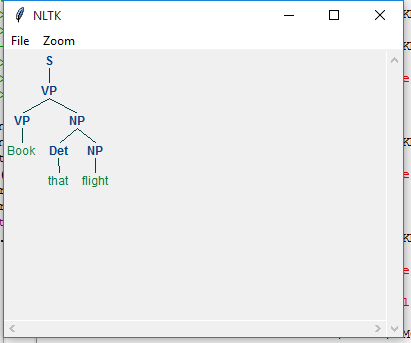
parser = nltk.ChartParser(grammar1)

for tree in parser.parse(all\_tokens):

 print(tree)

 tree.draw()

**Output:**



**b) Implementation of Deductive Chart Parsing using context free grammar and a given sentence.**

import nltk

from nltk import tokenize

grammar1 = nltk.CFG.fromstring("""

S -> NP VP

PP -> P NP

NP -> Det N | Det N PP | 'I'

VP -> V NP | VP PP

Det -> 'a' | 'my'

N -> 'bird' | 'balcony'

V -> 'saw'

P -> 'in'

""")

sentence = "I saw a bird in my balcony"

for index in range(len(sentence)):

 all\_tokens = tokenize.word\_tokenize(sentence)

print(all\_tokens)

# all\_tokens = ['I', 'saw', 'a', 'bird', 'in', 'my', 'balcony']

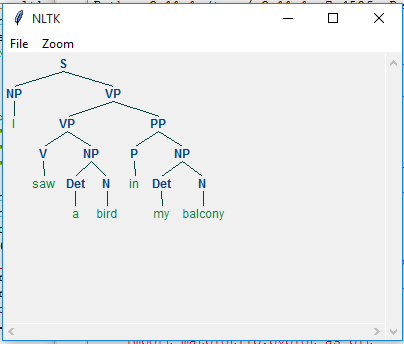
parser = nltk.ChartParser(grammar1)

for tree in parser.parse(all\_tokens):

 print(tree)

 tree.draw()

**Output:**



**Assignment No :- 06**

**Assignment Name:-** Write a program to Implement dependency parsing of a given text.

**Name: Nishikant Pawar** **Roll No:-**

**Class:-** **MSc CS Part 1** **Date:**

**Code:-**

#PARTICAL NO 5 OF NLP

import nltk

#nltk.download('punkt')

#nltk.download('averaged\_perceptron\_tagger')

from nltk import \*

# Example text

sample\_text = input("Enter the text:")

# Find all parts of speech in above sentence

tagged = pos\_tag(word\_tokenize(sample\_text))

#Extract all parts of speech from any text

chunker = RegexpParser("""

NP: {<DT>?<JJ>\*<NN>}

P: {<IN>}

V: {<V.\*>}

PP: {<p> <NP>}

VP: {<V> <NP|PP>\*}

""")

# Print all parts of speech in above sentence

output = chunker.parse(tagged)

print("After Extracting :\n", output)

# To draw the parse tree

output.draw()

**Output:-**

Enter the text:I am Neha from msc and I love to do yoga because it refresh your mind and soal and makes you more beautiful

After Extracting :

(S

I/PRP

(VP (V am/VBP))

Neha/RB

(P from/IN)

(NP msc/NN)

and/CC

I/PRP

(VP (V love/VBP))

to/TO

(VP (V do/VB))

yoga/RB

(P because/IN)

it/PRP

(VP (V refresh/VBZ))

your/PRP$

(NP mind/NN)

and/CC

(NP soal/NN)

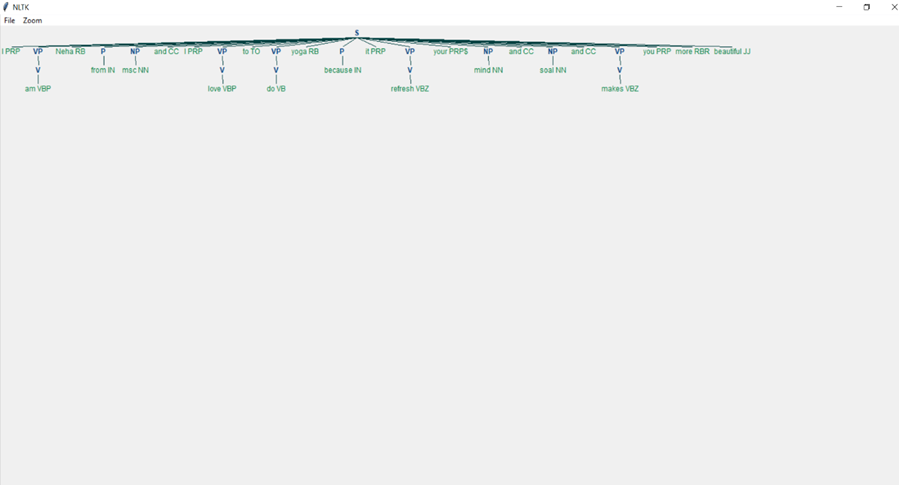
and/CC

(VP (V makes/VBZ))

you/PRP

more/RBR

beautiful/JJ)



**Assignment No :- 07**

**Assignment Name:-** Write a program to Implement Named Entity Recognition (NER)

**Name: Nishikant Pawar** **Roll No:-**

**Class:-** **MSc CS Part 1** **Date:**-

**Code:-**

# import modules and download packages

import nltk

#nltk.download('words')

#nltk.download('punkt')

#nltk.download('maxent\_ne\_chunker')

#nltk.download('averaged\_perceptron\_tagger')

#nltk.download('state\_union')

from nltk.corpus import state\_union

from nltk.tokenize import PunktSentenceTokenizer

# process the text and print Named entities

# tokenization

train\_text = state\_union.raw()

sample\_text = state\_union.raw("2006-GWBush.txt")

custom\_sent\_tokenizer = PunktSentenceTokenizer(train\_text)

tokenized = custom\_sent\_tokenizer.tokenize(sample\_text)

# function

def get\_named\_entity():

try:

for i in tokenized:

words = nltk.word\_tokenize(i)

tagged = nltk.pos\_tag(words)

namedEnt = nltk.ne\_chunk(tagged, binary=False)

namedEnt.draw()

except:

pass

get\_named\_entity()

**Output:**



**Assignment 08**

**Assignment Name: Write a program to implement text Summarization.**

**Name: Nishikant Pawar. Roll No: Class: MSc CS Part 1 Date:**

**---------------------------------------------------------------------------------------------------------------------**

**Source Code:**

import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize, sent\_tokenize

# Download necessary NLTK data

nltk.download('punkt')

nltk.download('stopwords')

# Input text - to summarize

text = (

"In an attempt to build an AI-ready workforce, Microsoft announced Intelligent Cloud Hub "

"which has been launched to empower the next generation of students with AI-ready skills. "

"Envisioned as a three-year collaborative program, Intelligent Cloud Hub will support around "

"100 institutions with AI infrastructure, course content and curriculum, developer support, "

"development tools and give students access to cloud and AI services. As part of the program, "

"the Redmond giant which wants to expand its reach and is planning to build a strong developer "

"ecosystem in India with the program will set up the core AI infrastructure and IoT Hub for the "

"selected campuses. The company will provide AI development tools and Azure AI services such as "

"Microsoft Cognitive Services, Bot Services and Azure Machine Learning."

)

# Preprocessing

stopWords = set(stopwords.words("english"))

words = word\_tokenize(text)

freqTable = dict()

# Creating frequency table for words

for word in words:

word = word.lower()

if word in stopWords:

continue

if word in freqTable:

freqTable[word] += 1

else:

freqTable[word] = 1

# Tokenizing sentences

sentences = sent\_tokenize(text)

sentenceValue = dict()

# Calculating sentence value based on word frequency

for sentence in sentences:

for word, freq in freqTable.items():

if word in sentence.lower():

if sentence in sentenceValue:

sentenceValue[sentence] += freq

else:

sentenceValue[sentence] = freq

# Calculate the average value of a sentence from the original text

sumValues = 0

for sentence in sentenceValue:

sumValues += sentenceValue[sentence]

# Avoid division by zero by ensuring there are sentences in the sentenceValue dictionary

if len(sentenceValue) > 0:

average = sumValues / len(sentenceValue)

else:

average = 0

# Generate summary

summary = ""

for sentence in sentences:

if (sentence in sentenceValue) and (sentenceValue[sentence] > (1.2 \* average)):

summary += " " + sentence

print(summary.strip())

**Output:**

Envisioned as a three-year collaborative program, Intelligent Cloud Hub will support around 100 institutions with AI infrastructure, course content and curriculum, developer support, development tools and give students access to cloud and AI services.