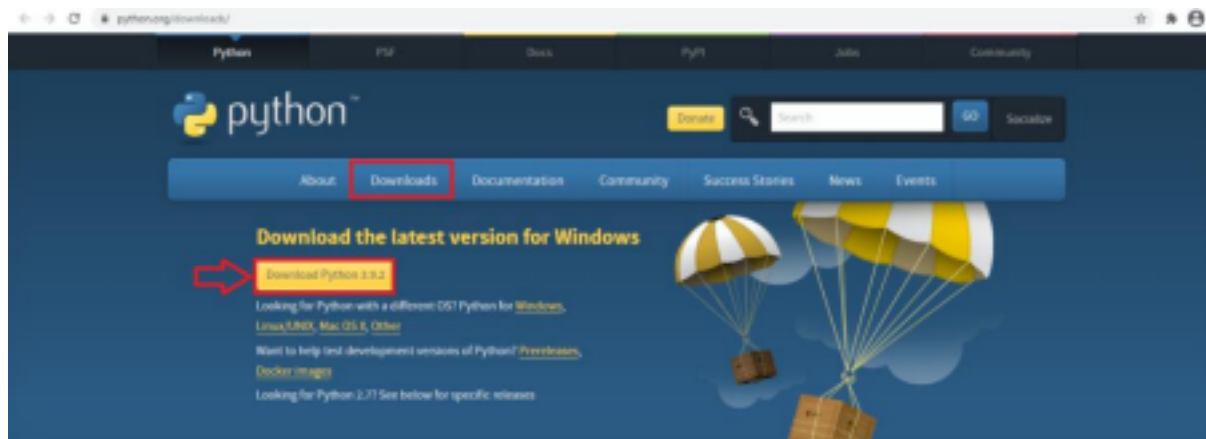


Practical No. 1

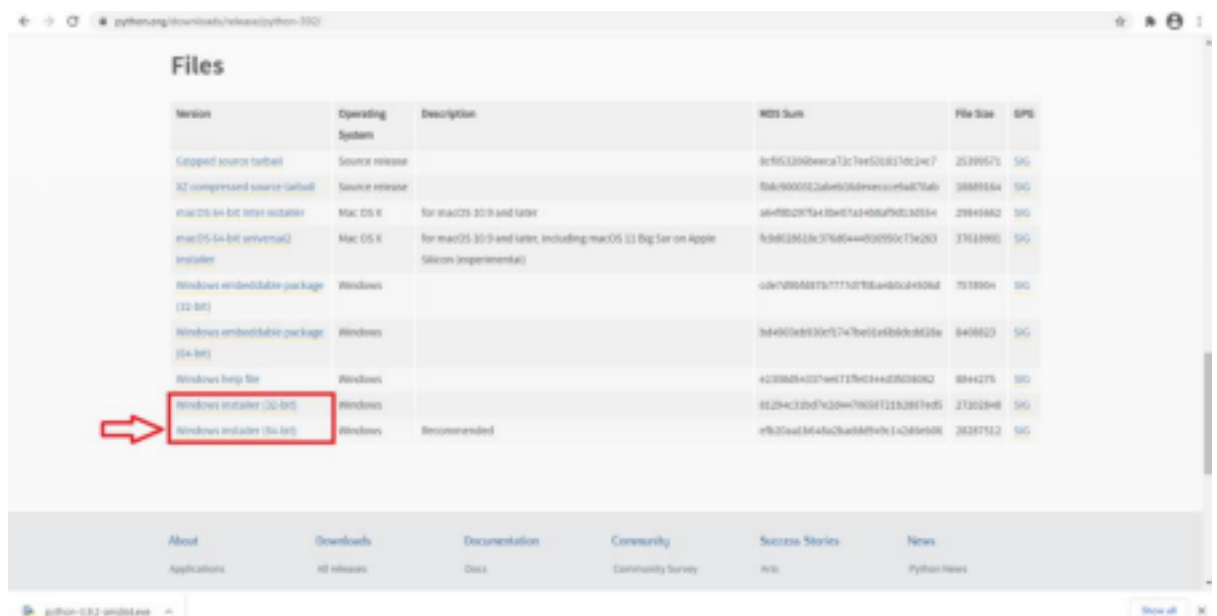
a) Install NLTK

Python 3.9.2 Installation on Windows

Step 1) Go to link <https://www.python.org/downloads/>, and select the latest version for windows.

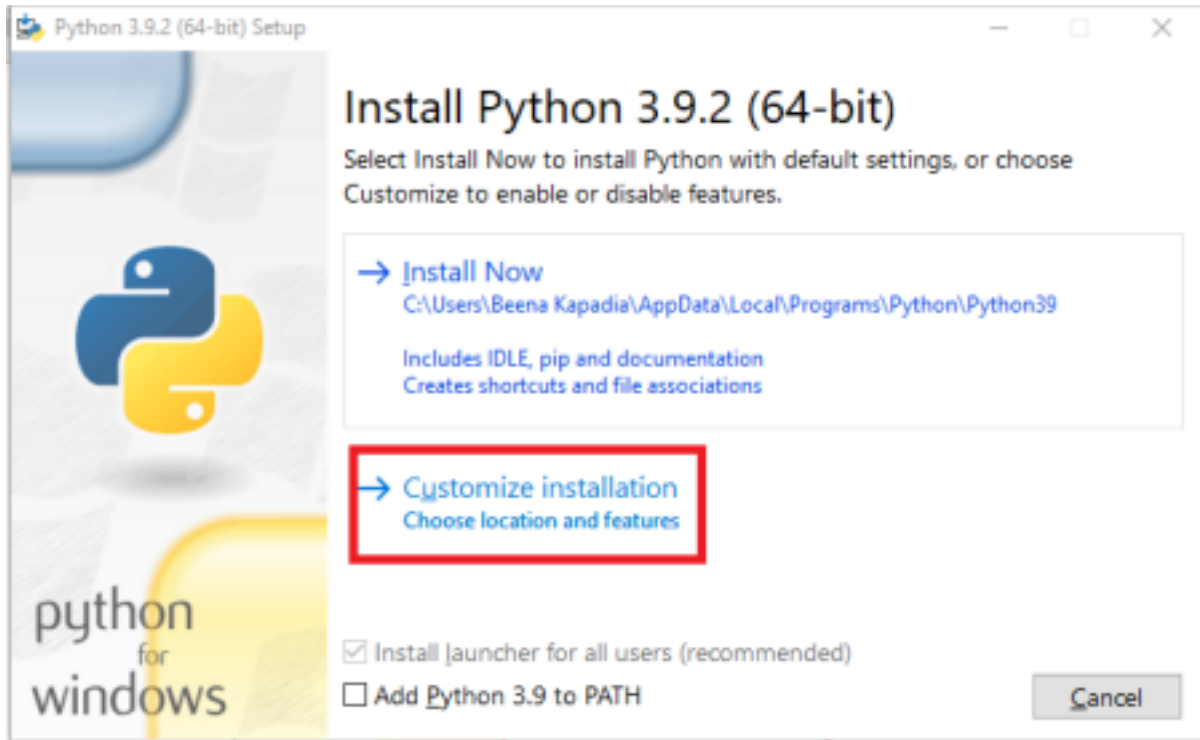


Note: If you don't want to download the latest version, you can visit the download tab and see all releases.

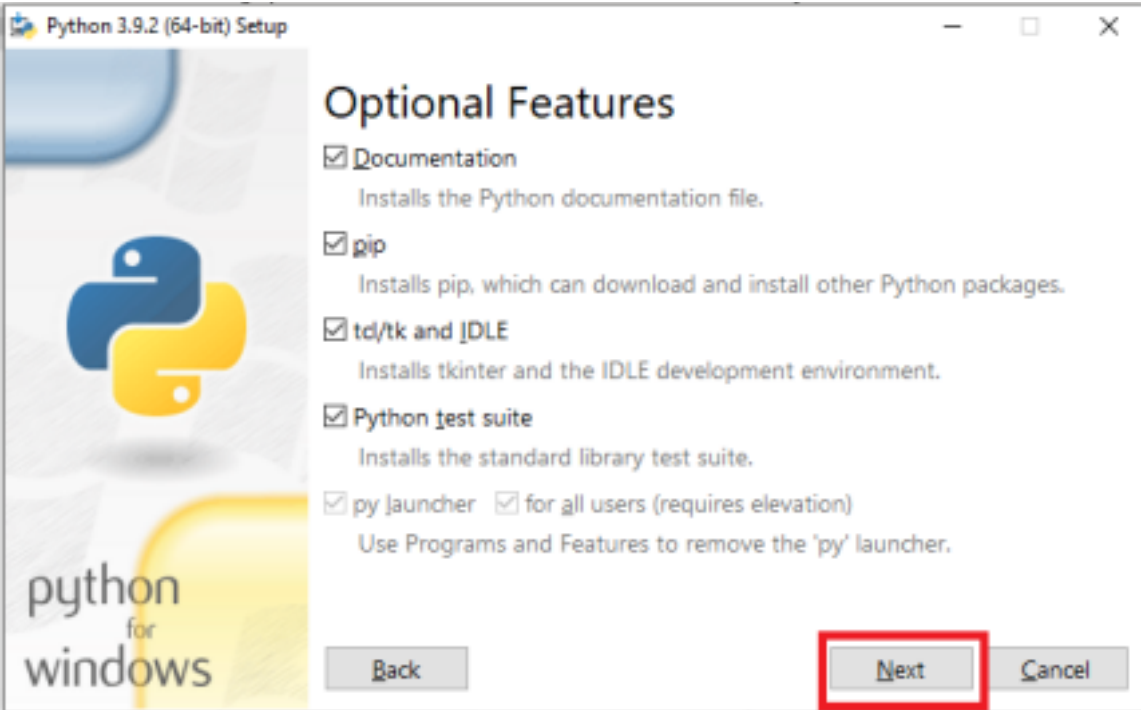


Step 2) Click on the Windows installer (64 bit)

Step 3) Select Customize Installation

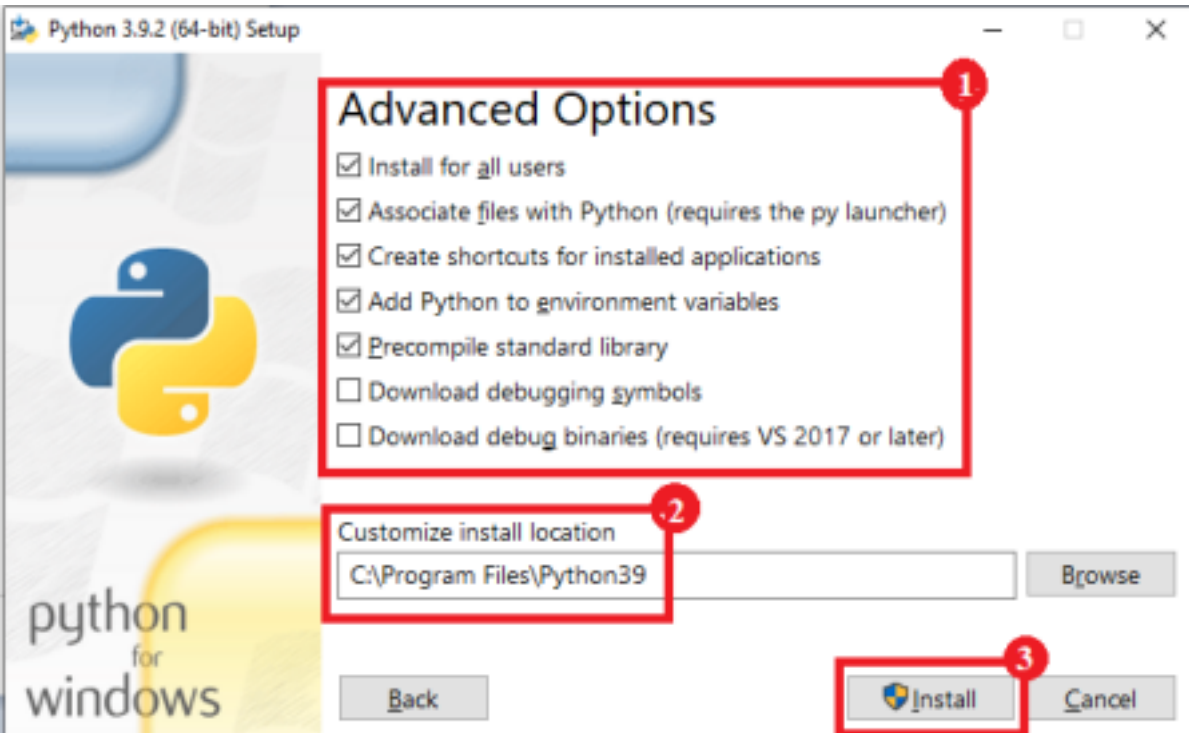


Step 4) Click NEXT



Step 5) In next screen

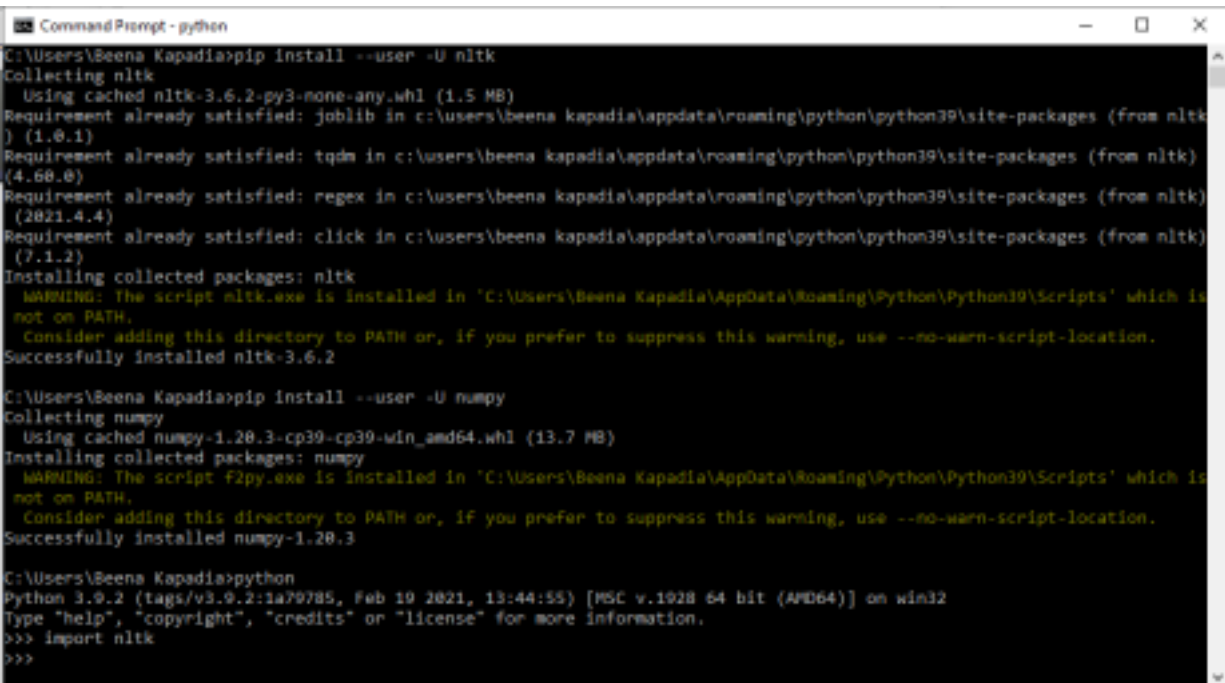
1. Select the advanced options
2. Give a Custom install location. Keep the default folder as c:\Program files\Python39
3. Click Install



Step 6) Click Close button once install is done.

Step 7) open command prompt window and run the following commands:

```
C:\Users\>pip install --upgrade pip
C:\Users\> pip install --user -U nltk
C:\Users\> pip install --user -U numpy
C:\Users\>python
>>> import nltk
>>>
```



```
Command Prompt - python
C:\Users\Beena Kapadia>pip install --user -U nltk
Collecting nltk
  Using cached nltk-3.6.2-py3-none-any.whl (1.5 MB)
Requirement already satisfied: joblib in c:\users\beena kapadia\appdata\roaming\python\python39\site-packages (from nltk) (1.0.1)
Requirement already satisfied: tqdm in c:\users\beena kapadia\appdata\roaming\python\python39\site-packages (from nltk) (4.60.0)
Requirement already satisfied: regex in c:\users\beena kapadia\appdata\roaming\python\python39\site-packages (from nltk) (2021.4.4)
Requirement already satisfied: click in c:\users\beena kapadia\appdata\roaming\python\python39\site-packages (from nltk) (7.1.2)
Installing collected packages: nltk
  WARNING: The script nltk.exe is installed in 'C:\Users\Beena Kapadia\AppData\Roaming\Python\Python39\Scripts' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed nltk-3.6.2

C:\Users\Beena Kapadia>pip install --user -U numpy
Collecting numpy
  Using cached numpy-1.20.3-cp39-cp39-win_amd64.whl (13.7 MB)
Installing collected packages: numpy
  WARNING: The script f2py.exe is installed in 'C:\Users\Beena Kapadia\AppData\Roaming\Python\Python39\Scripts' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed numpy-1.20.3

C:\Users\Beena Kapadia>python
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import nltk
>>>
```

a. Convert the given text to Speech

Import the required module for text

to speech conversion

!pip install gtts

from gtts import gTTS

This module is imported so that we can

```
# play the converted audio
import os

# The text that you want to convert to audio
mytext = 'Welcome to geeksforgeeks!'

# Language in which you want to convert
language = 'en'

# Passing the text and language to the engine,
# here we have marked slow=False. Which tells
# the module that the converted audio should
# have a high speed
myobj = gTTS(text=mytext, lang=language, slow=False)

# Saving the converted audio in a mp3 file named
# welcome
myobj.save("welcome.mp3")

# Playing the converted file
os.system("mpg321 welcome.mp3")
```

b) Convert the given text to speech.

Source code:

```
# Import the required module for text
# to speech conversion
from gtts import gTTS

# This module is imported so that we can
# play the converted audio
import os

# The text that you want to convert to audio
mytext = 'Welcome to geeksforgeeks!'

# Language in which you want to convert
language = 'en'

# Passing the text and language to the engine,
# here we have marked slow=False. Which tells
# the module that the converted audio should
# have a high speed
myobj = gTTS(text=mytext, lang=language, slow=False)

# Saving the converted audio in a mp3 file named
# welcome
myobj.save("welcome.mp3")

# Playing the converted file
os.system("welcome.mp3")
```

Practical 2

- a. Study of various Corpus – Brown, Inaugural, Reuters, udhr with various methods like fields, raw, words, sents, categories,

```
nltk.download('brown')

[nltk_data] Downloading package brown to
[nltk_data] C:\Users\lenovo\AppData\Roaming\nltk_data...
[nltk_data] Unzipping corpora\brown.zip.
True
```

```
import nltk
from nltk.corpus import brown
print('File ids of brown corpus\n',brown.fileids())
```

'''Let's pick out the first of these texts — Emma by Jane Austen — and give it a short name, emma, then find out how many words it contains:'''

```
ca01 = brown.words('ca01')
```

```
# display first few words
print('\nca01 has following words:\n',ca01)
```

```
# total number of words in ca01
print('\nca01 has',len(ca01),'words')
```

```
#categories or files
print('\n\nCategories or file in brown corpus:\n')
print(brown.categories())
```

'''display other information about each text, by looping over all the values of fileid corresponding to the brown file identifiers listed earlier and then computing statistics for each text.'''

```

print ('\n\nStatistics for each text:\n')
print
('AvgWordLen\tAvgSentenceLen\tno.ofTimesEachWordAppearsOnAvg\t\tFile
leName')
for fileid in brown.fileids():
    num_chars = len(brown.raw(fileid))
    num_words = len(brown.words(fileid))
    num_sents = len(brown.sents(fileid))
    num_vocab = len(set([w.lower() for w in brown.words(fileid)]))

    print (int(num_chars/num_words),'\t\t\t',
int(num_words/num_sents),'\t\t\t',
int(num_words/num_vocab),'\t\t\t', fileid)

```

b. Study Conditional frequency distributions

Conditional Frequency Distribution

Conditional Frequency Distribution is used when we want to count words meeting specific criteria satisfying a set of text.

```
import nltk
```

```
items = ['apple', 'apple', 'kiwi', 'cabbage', 'cabbage', 'potato']
```

```
nltk.FreqDist(items)
```

Output:

```
FreqDist({'apple': 2, 'cabbage': 2, 'kiwi': 1, 'potato': 1})
```

d. Study of tagged corpora with methods like tagged_sents, tagged_words.

Tokenization is used in natural language processing to split paragraphs and sentences into smaller units that can be more easily assigned meaning.

Word tokenization

```
import nltk

word_data = "It originated from the idea that there are readers who  
prefer learning new skills from the comforts of their drawing rooms"  
nltk_tokens = nltk.word_tokenize(word_data)  
print (nltk_tokens)
```

Tokenizing Sentences

We can also tokenize the sentences in a paragraph like we tokenized the words. We use the method `sent_tokenize` to achieve this. Below is an example.

```
import nltk

sentence_data = "Sun rises in the east. Sun sets in the west."  
nltk_tokens = nltk.sent_tokenize(sentence_data)  
print (nltk_tokens)
```

When we execute the above code, it produces the following result.

```
['Sun rises in the east.', 'Sun sets in the west.']
```

e. **Write a program to find the most frequent noun tags.**

Code:

```
import nltk

from collections import defaultdict

text = nltk.word_tokenize("Nick likes to play football. Nick does not like to  
play cricket.")
```

```

tagged = nltk.pos_tag(text)

print(tagged)

addNounWords = []

count=0

for words in tagged:

    val = tagged[count][1]

    if(val == 'NN' or val == 'NNS' or val == 'NNPS' or val == 'NNP'):

        addNounWords.append(tagged[count][0])

        count+=1

print ('the most common noun is ', addNounWords)

```

f. Map Words to Properties Using Python Dictionaries code:

[Python Dictionary - GeeksforGeeks](#)

```

#creating and printing a dictionary by mapping word with its properties
thisdict = {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964
}
print(thisdict)
print(thisdict["brand"])

```

```
print(len(thisdict))
print(type(thisdict))
```

g. Study i) DefaultTagger, ii) Regular expression tagger, iii) UnigramTagger

i) DefaultTagger

```
#Tagging a list of sentences
import nltk
from nltk.tag import DefaultTagger
exptagger = DefaultTagger('NN')
print(exptagger.tag_sents([['Hi', ','], ['How', 'are', 'you', '?']]))
```

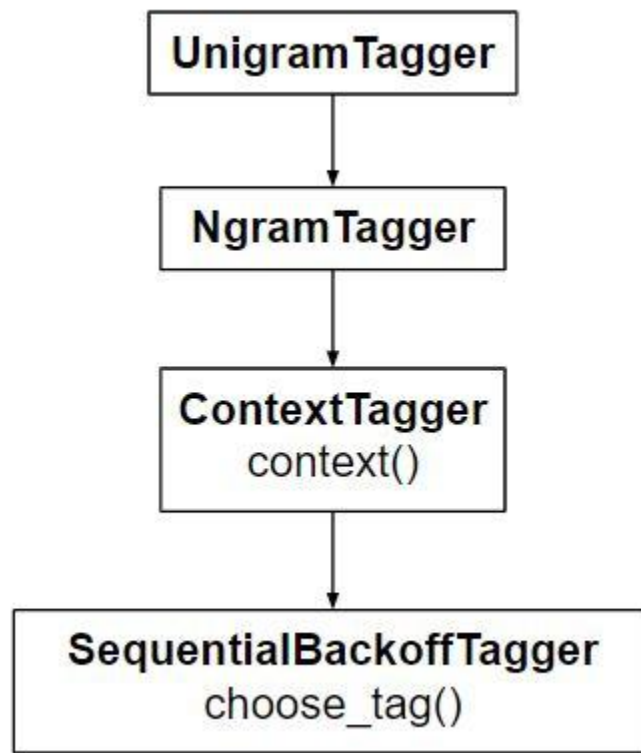
```
import re
```

```
txt = "The rain in Spain,Rain,Rain in Spain,Rain in Mumbai"
txt1="The rain in Spain"
txt2="The rain in spain"
x = re.search("^The.*Spain$", txt)
x1 = re.search("^The.*Spain$", txt1)
x2 = re.search("^The.*Spain$", txt2)
print(x)
print(x1)
print(x2)
```

Unigram tagger

A single token is referred to as a *Unigram*, for example – hello; movie; coding. This article is focused on **unigram tagger**. **Unigram Tagger**: For determining the Part of Speech tag, it only uses a single word. UnigramTagger inherits from NgramTagger, which is a subclass of

ContextTagger, which inherits from SequentialBackoffTagger. So, UnigramTagger is a single word context-based tagger.



```
from nltk.util import ngrams
```

```
n = 1
```

```
sentence = 'You will face many defeats in life, but never let yourself be  
defeated.'
```

```
unigrams = ngrams(sentence.split(), n)
```

```
for item in unigrams:
```

```
    print(item)
```

Practical No 3

- a. **Study of Wordnet Dictionary with methods as synsets, definitions, examples, antonyms**

WordNet is the lexical database i.e. dictionary for the English language, specifically designed for natural language processing.

Synset is a special kind of a simple interface that is present in NLTK to look up words in WordNet.

WordNet® is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations

```
from nltk.corpus import wordnet
```

```
syn = wordnet.synsets('hello')[0]
```

```
print ("Synset name : ", syn.name())
```

```
# Defining the word
```

```
print ("\nSynset meaning : ", syn.definition())
```

```
# list of phrases that use the word in context
```

```
print ("\nSynset example : ", syn.examples())
```

```
print(wordnet.lemma('buy.v.01.buy').antonyms())
```

Study lemmas, hyponyms, hypernyms.

Hyponymy and hyponyms

Hyponymy describes the relationship between a generic term and instances of the specified generic term. Here, a generic term is called a hypernym, and instances of the generic term are called hyponyms.

So, color is a hypernym; red, green, yellow, and so on are hyponyms.

```
from nltk.corpus import wordnet
```

```
My_sysn = wordnet.synsets("Plane")[0]
```

```
print("Print just the name:", My_sysn.name())
```

```
print("The Hypernym for the word is:",My_sysn.hypernyms(),'\n')
```

```
print("The Hyponyms for the word is:",My_sysn.hyponyms())
```

Lemma:

Original Word ---> Root Word (lemma) Feature

meeting ---> meet (core-word extraction)

was ---> be (tense conversion to present tense)

mice ---> mouse (plural to singular)

```
import nltk
```

```
nltk.download('wordnet')
```

```
from nltk.stem import WordNetLemmatizer
```

```
# Create WordNetLemmatizer object
```

```
wnl = WordNetLemmatizer()
```

```
# single word lemmatization examples
```

```
list1 = ['kites', 'babies', 'dogs', 'flying', 'smiling',  
         'driving', 'died', 'tried', 'feet']
```

```
for words in list1:
```

```
    print(words + " ---> " + wnl.lemmatize(words))
```

- a. Write a program using python to find synonym and antonym of word "active" using Wordnet.

```
import nltk
```

```

nltk.download('wordnet')
from nltk.corpus import wordnet
synonyms = []
antonyms = []

for syn in wordnet.synsets("active"):
    for l in syn.lemmas():
        synonyms.append(l.name())
    if l.antonyms():
        antonyms.append(l.antonyms()[0].name())
print(set(synonyms))
print(set(antonyms))

```

Compare two nouns

```

import nltk
from nltk.corpus import wordnet
syn1 = wordnet.synsets('football')
syn2 = wordnet.synsets('soccer')
# A word may have multiple synsets, so need to compare each synset of
word1 with synset of word2
for s1 in syn1:
    for s2 in syn2:
        print("Path similarity of: ")
        print(s1, '(', s1.pos(), ')', '[', s1.definition(), ']')

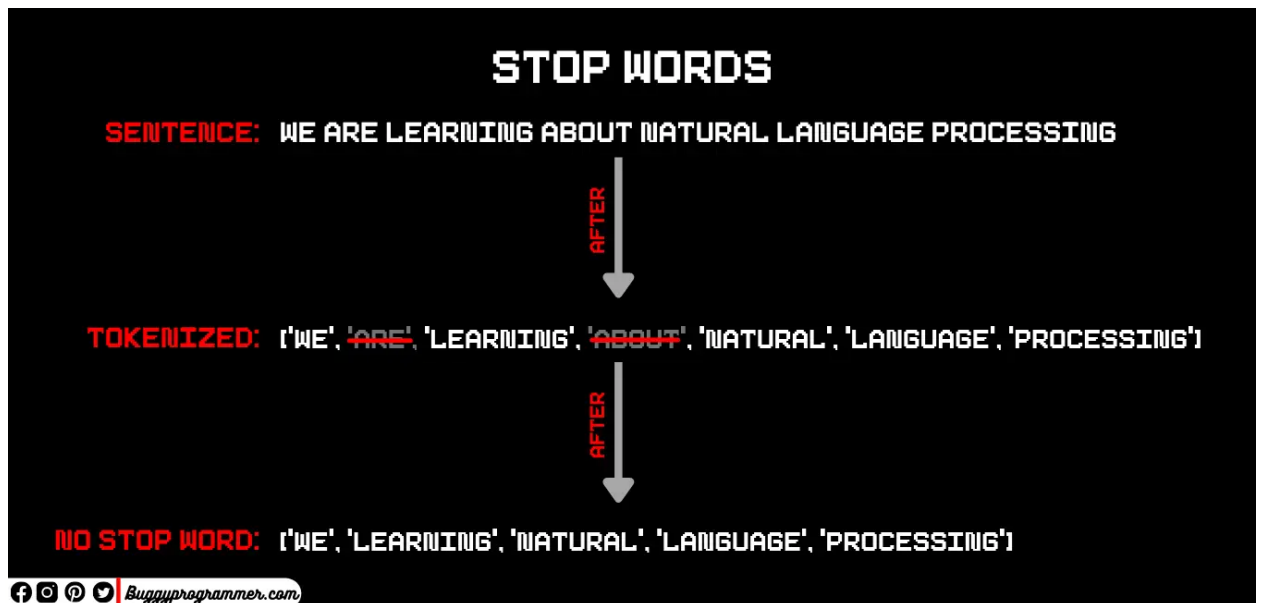
```



```
print(s2, '(', s2.pos(), ')', '[', s2.definition(), ']')  
print(" is", s1.path_similarity(s2))  
print()
```

Handling stopwords:

i) Using nltk Adding or Removing Stop Words in NLTK's Default Stop Word List



Step1 : word tokenize

Stopt word call

Add

remove

import nltk

nltk.download('punkt')

from nltk.corpus import stopwords

nltk.download('stopwords')

from nltk.tokenize import word_tokenize

text = "Yashesh likes to play football, however he is not too fond of tennis."

text_tokens = word_tokenize(text)

print(text_tokens)

tokens_without_sw = [word for word in text_tokens if not word in

stopwords.words()]

```
print(tokens_without_sw)
```

```
#add the word play to the NLTK stop word collection
```

```
all_stopwords = stopwords.words('english')
```

```
all_stopwords.append('play')
```

```
text_tokens = word_tokenize(text)
```

```
tokens_without_sw = [word for word in text_tokens if not word in  
all_stopwords]
```

```
print(tokens_without_sw)
```

```
#remove 'not' from stop word collection
```

```
all_stopwords.remove('not')
```

```
text_tokens = word_tokenize(text)
```

```
tokens_without_sw = [word for word in text_tokens if not word in  
all_stopwords]
```

```
print(tokens_without_sw)
```

**Using Gensim Adding and Removing Stop Words in
Default Gensim Stop Words List**

Practical 4

a. Tokenization using Python's split() function

```
text = """ This tool is an a beta stage. Alexa developers can use Get  
Metrics API to seamlessly analyse metric. It also supports custom skill  
model, prebuilt Flash Briefing model, and the Smart Home Skill API.  
You can use this tool for creation of monitors, alarms, and dashboards  
that spotlight changes. The release of these three tools will enable  
developers to create visual rich skills for Alexa devices with screens.  
Amazon describes these tools as the collection of tech and tools for  
creating visually rich and interactive voice experiences. """
```

```
data = text.split('.')
```

```
for i in data:
```

```
    print (i)
```

b. Tokenization using Regular Expressions (RegEx)

```
import nltk
```

```
# import RegexpTokenizer() method from nltk

from nltk.tokenize import RegexpTokenizer

# Create a reference variable for Class RegexpTokenizer

tk = RegexpTokenizer('\s+', gaps = True)

# Create a string input

str = "I love to study Natural Language Processing in Python"

# Use tokenize method

tokens = tk.tokenize(str)

print(tokens)
```

c. Tokenization using NLTK

```
import nltk
```

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

```
from nltk.tokenize import word_tokenize
```

```
# Create a string input
```

```
str = "I love to study Natural Language Processing in Python"
```

```
# Use tokenize method
```

```
print(word_tokenize(str))
```

d. Tokenization using the spaCy library

```
import spacy
```

```
nlp = spacy.blank("en")
```

```
print(nlp)
```

```
# Create a string input
```

```
str = "I love to study Natural Language Processing in Python"
```

```
# Create an instance of document;
```

```
# doc object is a container for a sequence of Token objects.
```

```
doc = nlp(str)
```

```
# Read the words; Print the words
```

```
#
```

```
words = [word.text for word in doc]
```

```
print(words)
```

Tokenization using Keras

Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow.

- Simple -- but not simplistic. Keras reduces developer *cognitive load* to free you to focus on the parts of the problem that really matter.
- Flexible -- Keras adopts the principle of *progressive disclosure of complexity*: simple workflows should be quick and easy, while arbitrarily advanced workflows should be *possible* via a clear path that builds upon what you've already learned.
- Powerful -- Keras provides industry-strength performance and scalability: it is used by organizations and companies including NASA, YouTube, or Waymo.
-

```
import keras
```

```
from keras.preprocessing.text import text_to_word_sequence
```

```
# Create a string input
```

```
str = "I love to study Natural Language Processing in Python"
```

```
# tokenizing the text
```

```
tokens = text_to_word_sequence(str)
```

```
print(tokens)
```

Tokenization using Gensim


```
#pip install gensim
```

```
from gensim.utils import tokenize
```

```
# Create a string input
```

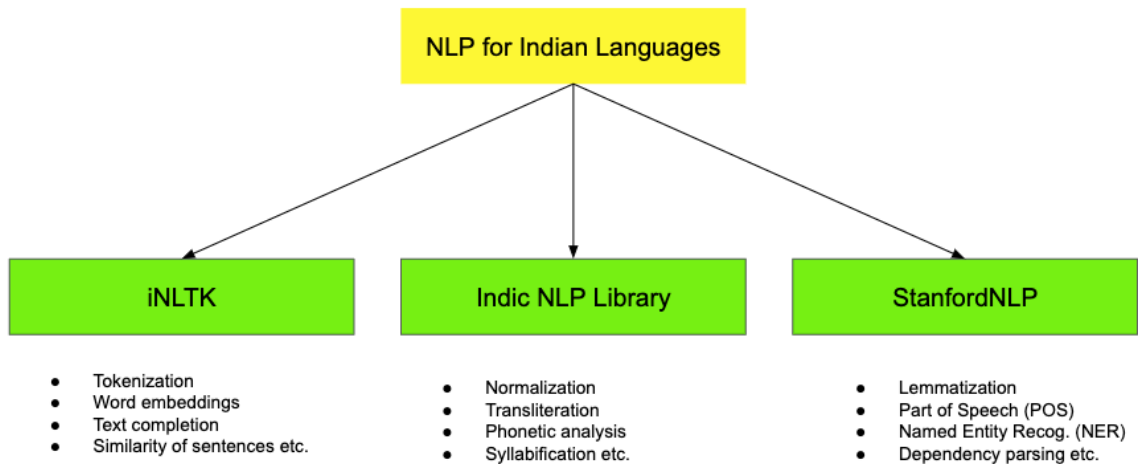
```
str = "I love to study Natural Language Processing in Python"
```

```
# tokenizing the text
```

```
list(tokenize(str))
```

Practical 5

Import NLP Libraries for Indian Languages and perform:



What is PyTorch used for?



PyTorch is a fully featured framework for building deep learning models, which is a type of machine learning that's commonly used in applications like image recognition and language processing.

pip install torch==1.7.1+cpu -f

https://download.pytorch.org/whl/torch_stable.html

<https://www.analyticsvidhya.com/blog/2020/01/3-important-nlp-libraries-indian-languages-python/>

a.

[NLP Libraries For Indian Languages | NLP For Indian Languages \(analyticsvidhya.com\)](#)

Practical 6

Illustrate part of speech tagging.

- a. Part of speech Tagging and chunking of user defined text.**
- b. Named Entity recognition of user defined text.**
- c. Named Entity recognition with diagram using NLTK corpus – treebank**

a. Part of speech Tagging and chunking of user defined text.

Chunking is the process of grouping similar words together based on the nature of the word. In the below example we define a grammar by which the chunk must be generated. The grammar suggests the sequence of the phrases like nouns and adj...

```
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)

```

b. Named Entity recognition using user defined text.

```
!pip install -U spacy
```

```
!python -m spacy download en_core_web_sm
```

```
import spacy
```

```
# Load English tokenizer, tagger, parser and NER
```

```
nlp = spacy.load("en_core_web_sm")
```

```
# Process whole documents
```

```
text = ("When Sebastian Thrun started working on self-driving cars at "
"Google in 2007, few people outside of the company took him " "seriously. "I
```

can tell you very senior CEOs of major American " "car companies would shake my hand and turn away because I wasn't " "worth talking to," said Thrun, in an interview with Recode earlier " "this week.")

```
doc = nlp(text)
```

```
# Analyse syntax
```

```
print("Noun phrases:", [chunk.text for chunk in doc.noun_chunks])
```

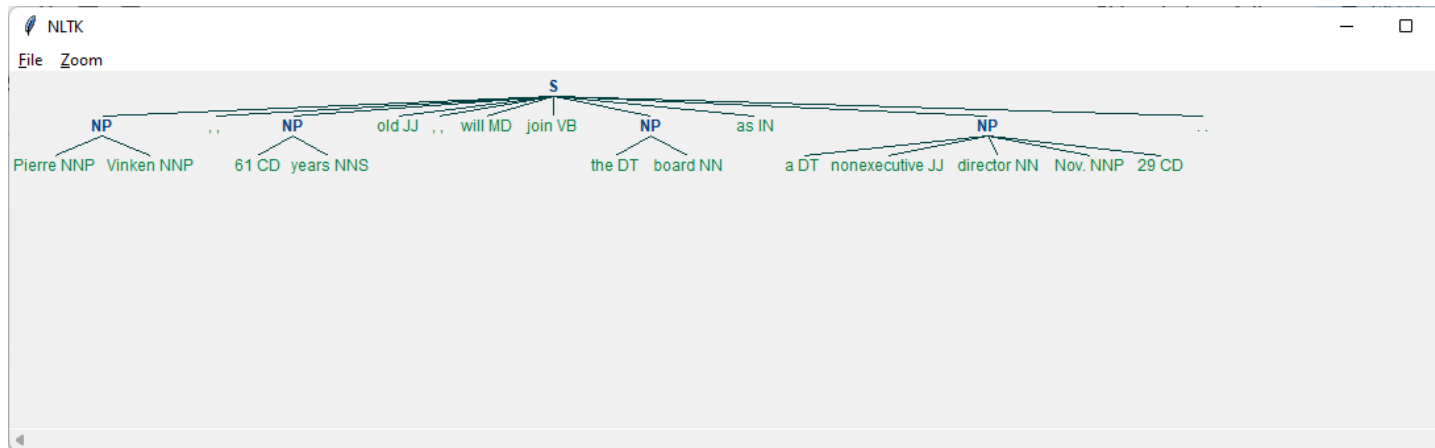
```
print("Verbs:", [token.lemma_ for token in doc if token.pos_ == "VERB"])
```

c. Named Entity recognition with diagram using NLTK corpus – treebank. Source code:

Note: It runs on Python IDLE

```
import nltk
nltk.download('treebank')
from nltk.corpus import treebank_chunk
treebank_chunk.tagged_sents()[0]

treebank_chunk.chunked_sents()[0]
treebank_chunk.chunked_sents()[0].draw()
```



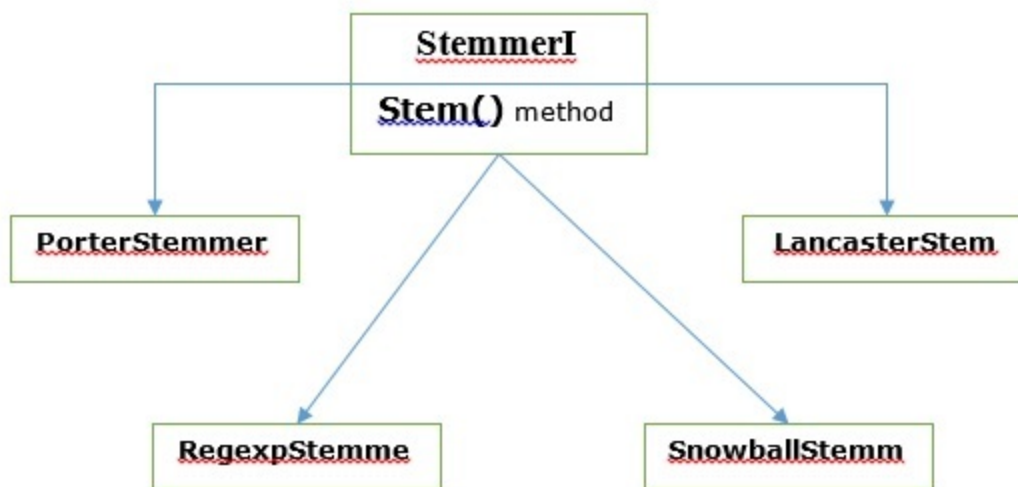
Practical no 7

Finite state automata

- a) Define grammar using nltk. Analyze a sentence using the same.

Practical 8

Study PorterStemmer, LancasterStemmer, RegexpStemmer, SnowballStemmer Study WordNetLemmatizer



a. PorterStemmer

stemming is the process of reducing a word to its word stem. Word stem is a base or root form of the word and doesn't need to be an existing word. For example, the Porter algorithm reduces the words “argue”, “argued”, “argues” and “arguing” to the stem “argu” which isn't an existing word.

```
import nltk
from nltk.stem import PorterStemmer
word_stemmer = PorterStemmer()
print(word_stemmer.stem('writing'))
```

b. LancasterStemmer

the Lancaster stemmer consists of a set of rules where each rule specifies either deletion or replacement of an ending.

Example

the words “programming,” “programmer,” and “programs” can all be reduced down to the common word stem “program.”

```
import nltk
from nltk.stem import LancasterStemmer
Lanc_stemmer = LancasterStemmer()
print(Lanc_stemmer.stem('writing'))
```

c. #RegexStemmer

It basically takes a single regular expression and removes any prefix or suffix that matches the expression.

```
import nltk
from nltk.stem import RegexStemmer
Reg_stemmer = RegexStemmer('ing$|s$|e$|able$', min=4)
print(Reg_stemmer.stem('writing'))
```

Snowball stemming algorithm

Snowball is a small string processing language for creating stemming algorithms for use in Information Retrieval, plus a collection of stemming algorithms implemented using it.

Snowball Stemmer: It is a stemming algorithm which is also known as the Porter2 stemming algorithm as it is a better version of the Porter Stemmer since some issues of it were fixed in this stemmer.

```
import nltk
from nltk.stem import SnowballStemmer
english_stemmer = SnowballStemmer('english')
print(english_stemmer.stem('writing'))
```

#WordNetLemmatizer

```
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet

lemmatizer = WordNetLemmatizer()

print("love :", lemmatizer.lemmatize("loves", wordnet.VERB))
print("loving :", lemmatizer.lemmatize("loving", wordnet.VERB))
print("loved :", lemmatizer.lemmatize("loved", pos=wordnet.VERB))
```

Practical no 9

<https://hands-on.cloud/naive-bayes-classifier-python-tutorial/>

```
def bayesTheorem(pA, pB, pBA):
```

```
    return pA * pBA / pB
```

```
#define probabilities
```

```
pRain = 0.2
```

```
pCloudy = 0.4
```

```
pCloudyRain = 0.85
```

```
#use function to calculate conditional probability
```

```
bayesTheorem(pRain, pCloudy, pCloudyRain)
```

Or

```
# load the iris dataset
```

```
from sklearn.datasets import load_iris
```

```
iris = load_iris()
```

```
# store the feature matrix (X) and response vector (y)
```

```
X = iris.data
```

```
y = iris.target
```

```
# splitting X and y into training and testing sets
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
```

```
random_state=1)
```

```
# training the model on training set
```

```
from sklearn.naive_bayes import GaussianNB
```

```

gnb = GaussianNB()
gnb.fit(X_train, y_train)

# making predictions on the testing set
y_pred = gnb.predict(X_test)

# comparing actual response values (y_test) with predicted response
values (y_pred)
from sklearn import metrics
print("Gaussian Naive Bayes model accuracy(in %):",
metrics.accuracy_score(y_test, y_pred)*100)
https://www.geeksforgeeks.org/naive-bayes-classifiers/

```

Practical 11

a) Multiword Expressions in NLP

```

import nltk
from nltk.tokenize import MWETokenizer
nltk.download('punkt')
from nltk import sent_tokenize, word_tokenize
s = "'Good cake cost Rs.1500\kg in Mumbai.Please buy me one of
them.\n\nThanks.'"
mwe = MWETokenizer([( 'New', 'York'), ('Hong', 'Kong')],
separator='_')
for sent in sent_tokenize(s):
    print(mwe.tokenize(word_tokenize(sent)))

```

b) Normalized Web Distance and Word Similarity
Source code: