TABLE OF CONTENTS

Lab 9

Aim- Write a program for tokenization of word and sentence using NLTK package in python.

Also perform:

1. Stop word removal
2. Stemming
3. Lemmatization
4. POS tagging (parsing)
5. Parsing of a sentence

Theory

The Natural Language Toolkit (NLTK) is a popular open-source Python library used for natural language processing (NLP) tasks such as tokenization, stemming, lemmatization, part-of-speech tagging, named entity recognition, parsing, and semantic analysis. NLTK provides a comprehensive set of tools and resources for processing and analysing human language data.

Tokenization is essentially splitting a phrase, sentence, paragraph, or an entire text document into smaller units, such as individual words or terms. Each of these smaller units are called tokens. The tokens could be words, numbers or punctuation marks. In tokenization, smaller units are created by locating word boundaries. These are the ending point of a word and the beginning of the next word.

Stop words are commonly used words that are filtered out from natural language processing tasks like text analysis and information retrieval. These words are considered insignificant and do not add meaning to the content of a document or sentence. Stop words typically include pronouns, prepositions, conjunctions, and other frequently occurring words in a language, such as "the", "and", "a", "an", "in", "on", "of", "to", etc.

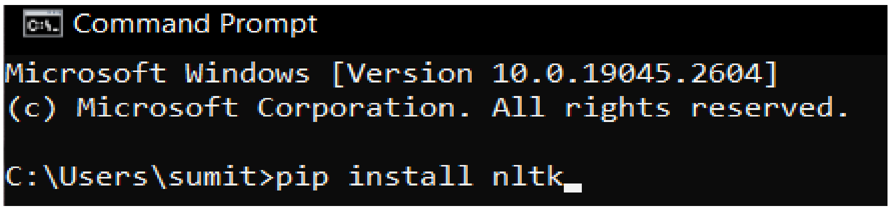
Stemming is a text processing task in which words can be reduce to their root, which is the core part of a word. For example, the words “helping” and “helper” share the root “help.” Stemming allows you to zero in on the basic meaning of a word rather than all the details of how it’s being used. Types of stemming:

1. **Understemming** happens when two related words should be reduced to the same stem but aren’t. This is a false negative.
2. **Overstemming** happens when two unrelated words are reduced to the same stem even though they shouldn’t be. This is a false positive.

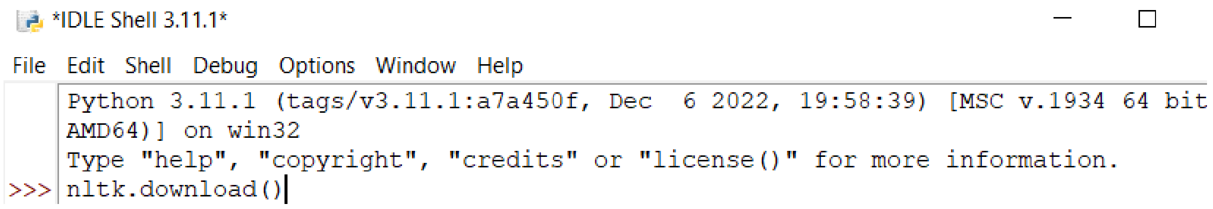
Lemmatization reduces words to their core meaning, but it will give a complete English word that makes sense on its own instead of just a fragment of a word like 'discoveri'.

Lemma is a word that represents a whole group of words, and that group of words is called a **lexeme**. For example, if you were to look up the word “blending” in a dictionary, then you’d need to look at the entry for “blend,” but you would find “blending” listed in that entry. In this example, “blend” is the lemma, and “blending” is part of the lexeme.

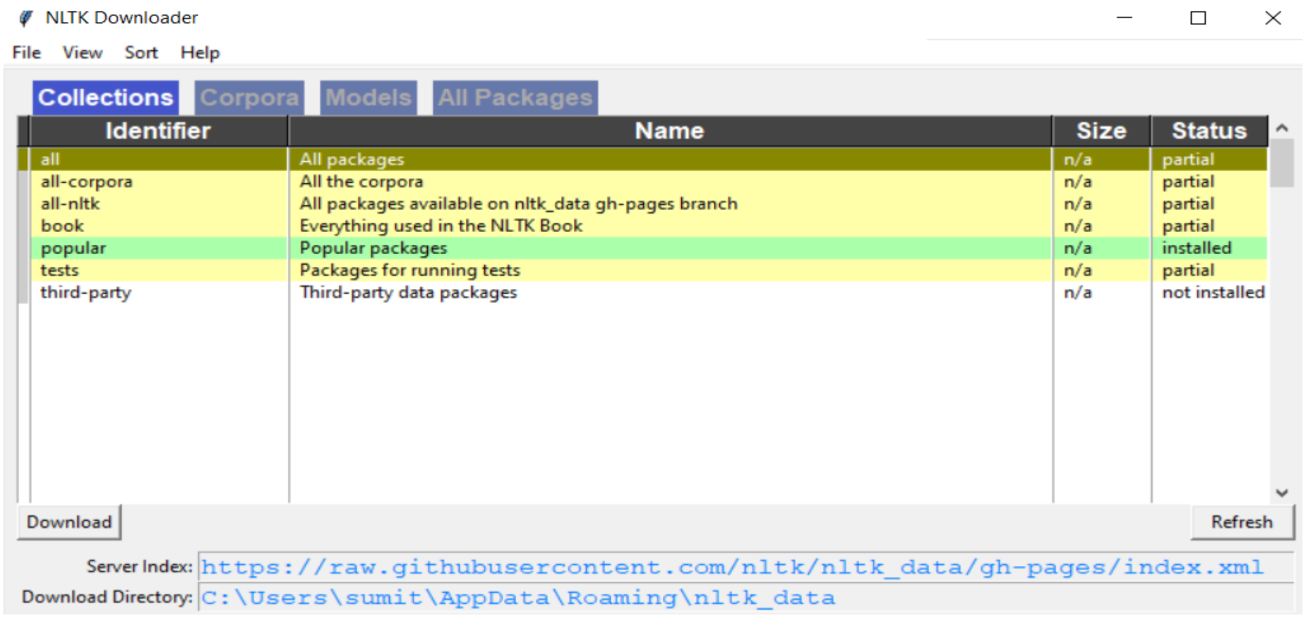
Parts of Speech Tagging(POS) is a process to mark up the words in text format for a particular part of a speech based on its definition and context. It is responsible for text reading in a language and assigning some specific token (Parts of Speech) to each word. It is also called grammatical tagging.



```
Command Prompt
Microsoft Windows [Version 10.0.19045.2604]
(c) Microsoft Corporation. All rights reserved.

C:\Users\sumit>pip install nltk_
```



```
*IDLE Shell 3.11.1*
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> nltk.download()
```



Identifier	Name	Size	Status
all	All packages	n/a	partial
all-corpora	All the corpora	n/a	partial
all-nltk	All packages available on nltk_data gh-pages branch	n/a	partial
book	Everything used in the NLTK Book	n/a	partial
popular	Popular packages	n/a	installed
tests	Packages for running tests	n/a	partial
third-party	Third-party data packages	n/a	not installed

Server Index: https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml
Download Directory: C:\Users\sumit\AppData\Roaming\nltk_data

Code

#Code for Tokenization of words and sentence

import nltk

from nltk.tokenize import sent_tokenize, word_tokenize

text = "Cryptarithmetic Problem is a type of constraint satisfaction problem where the game is about digits and its unique replacement " + \

"either with alphabets or other symbols. In cryptarithmetic problem, the digits (0-9) get substituted by some possible alphabets or symbols. " + \

"The task in cryptarithmetic problem is to substitute each digit with an alphabet to get the result arithmetically correct. "

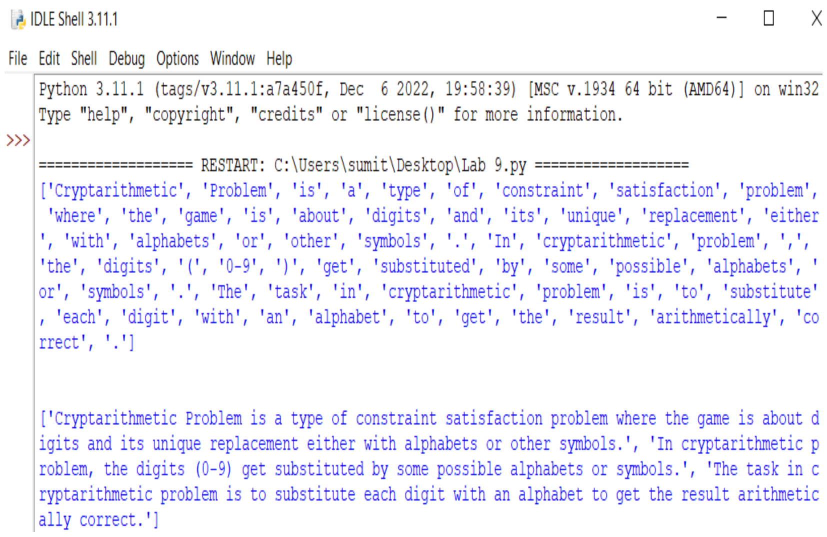
nltk_tokens = nltk.sent_tokenize(text)

print(word_tokenize(text))

print("\n")

print(sent_tokenize(text))

Output



```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: C:\Users\sumit\Desktop\Lab 9.py ==================
['Cryptarithmetic', 'Problem', 'is', 'a', 'type', 'of', 'constraint', 'satisfaction', 'problem',
 'where', 'the', 'game', 'is', 'about', 'digits', 'and', 'its', 'unique', 'replacement', 'either
', 'with', 'alphabets', 'or', 'other', 'symbols', '.', 'In', 'cryptarithmetic', 'problem', ',',
'the', 'digits', '(', '0-9', ')', 'get', 'substituted', 'by', 'some', 'possible', 'alphabets', '
or', 'symbols', '.', 'The', 'task', 'in', 'cryptarithmetic', 'problem', 'is', 'to', 'substitute'
, 'each', 'digit', 'with', 'an', 'alphabet', 'to', 'get', 'the', 'result', 'arithmetically', 'co
rrect', '.']


['Cryptarithmetic Problem is a type of constraint satisfaction problem where the game is about d
igits and its unique replacement either with alphabets or other symbols.', 'In cryptarithmetic p
roblem, the digits (0-9) get substituted by some possible alphabets or symbols.', 'The task in c
ryptarithmetic problem is to substitute each digit with an alphabet to get the result arithmetic
ally correct.']
```

#Code for stop word removal

import nltk

from nltk.corpus import stopwords

nltk.download('stopwords')

from nltk.tokenize import word_tokenize

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

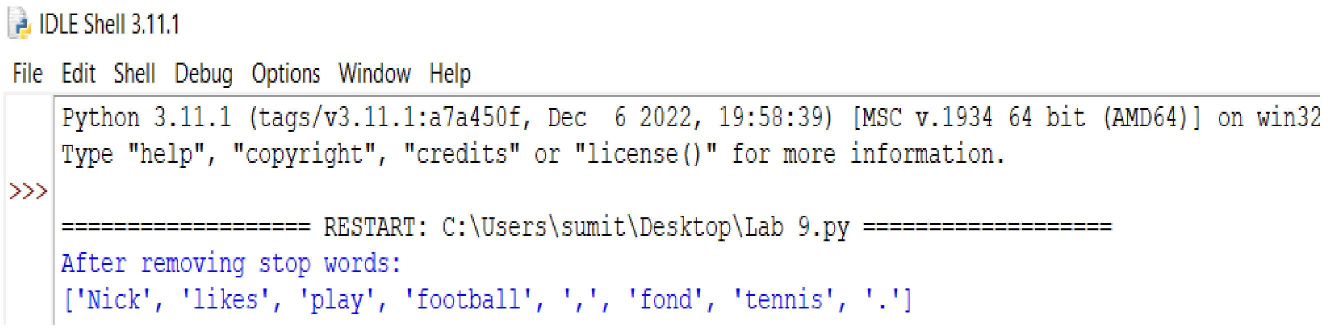
text_tokens = word_tokenize(text)

tokens_without_sw = [word for word in text_tokens if not word in stopwords.words()]

print("After removing stop words:")

print(tokens_without_sw)

Output



```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: C:\Users\sumit\Desktop\Lab 9.py ===================
After removing stop words:
['Nick', 'likes', 'play', 'football', ',', 'fond', 'tennis', '.']
```

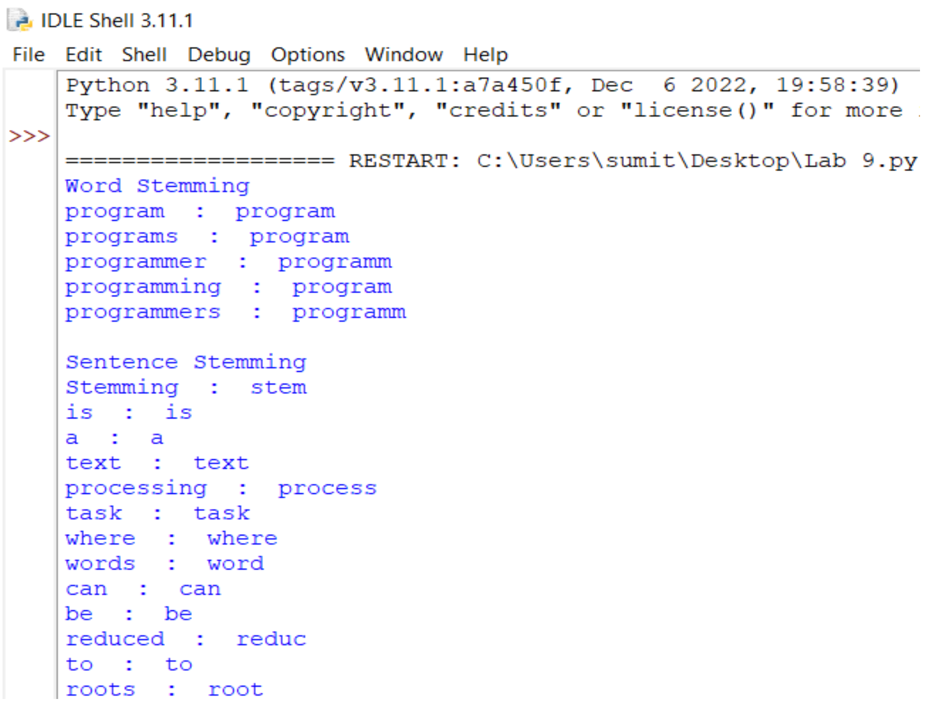
#Code for stemming

from nltk.stem import PorterStemmer

from nltk.tokenize import word_tokenize

```
ps = PorterStemmer()
words = ["program", "programs", "programmer", "programming", "programmers"]
sentence = "Stemming is a text processing task where words can be reduced to roots"
print("\nWord Stemming")
for w in words:
        print(w, " : ", ps.stem(w))
print("\nSentence Stemming")
words = word_tokenize(sentence)
for w in words:
  print(w, " : ", ps.stem(w))
```

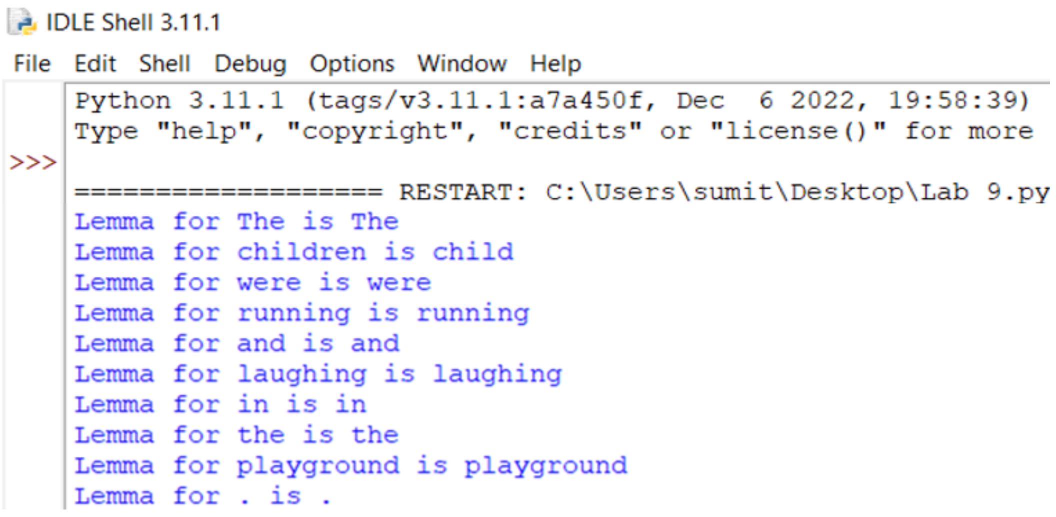
Output



```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39)
Type "help", "copyright", "credits" or "license()" for more
>>>
=================== RESTART: C:\Users\sumit\Desktop\Lab 9.py
Word Stemming
program  :  program
programs  :  program
programmer  :  programm
programming  :  program
programmers  :  programm

Sentence Stemming
Stemming  :  stem
is  :  is
a  :  a
text  :  text
processing  :  process
task  :  task
where  :  where
words  :  word
can  :  can
be  :  be
reduced  :  reduc
to  :  to
roots  :  root
```

```
#Code for Lemmatization
import nltk
from nltk.stem import WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()
text = "The children were running and laughing in the playground."
tokenization = nltk.word_tokenize(text)
print("\n")
for w in tokenization:
  print("Lemma for {} is {}".format(w, wordnet_lemmatizer.lemmatize(w)))
```

Output



```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39)
Type "help", "copyright", "credits" or "license()" for more
>>>
==================== RESTART: C:\Users\sumit\Desktop\Lab 9.py
Lemma for The is The
Lemma for children is child
Lemma for were is were
Lemma for running is running
Lemma for and is and
Lemma for laughing is laughing
Lemma for in is in
Lemma for the is the
Lemma for playground is playground
Lemma for . is .
```

```
#Code for POS tagging
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize
stop_words = set(stopwords.words('english'))
txt ="Parts of Speech Tagging is a process to mark up the words in text " \
    "format for a particular part of a speech based on its definition " \
    "It is responsible for text reading in a language " \
    "It is also called grammatical tagging. "
tokenized = sent_tokenize(txt)
for i in tokenized:
  wordsList = nltk.word_tokenize(i)
#Using a POS Tagger.
tagged = nltk.pos_tag(wordsList)
print("\nPOS tagged text:")
print(tagged)
```

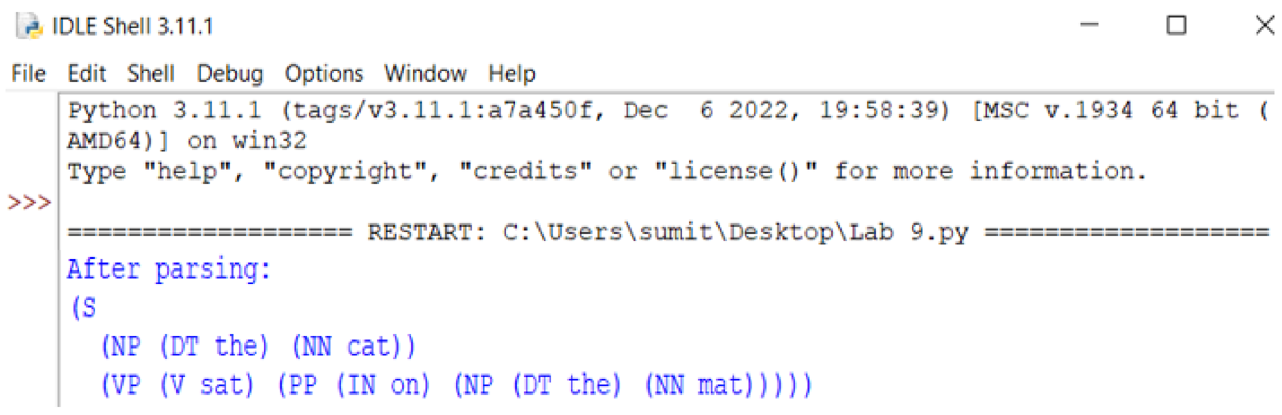
Output

```
POS tagged text:
[('Parts', 'NNS'), ('of', 'IN'), ('Speech', 'NNP'), ('Tagging', 'NNP'), ('is', 'VBZ'), ('
a', 'DT'), ('process', 'NN'), ('to', 'TO'), ('mark', 'VB'), ('up', 'RP'), ('the', 'DT'),
('words', 'NNS'), ('in', 'IN'), ('text', 'JJ'), ('format', 'NN'), ('for', 'IN'), ('a', 'D
T'), ('particular', 'JJ'), ('part', 'NN'), ('of', 'IN'), ('a', 'DT'), ('speech', 'NN'), (
'based', 'VBN'), ('on', 'IN'), ('its', 'PRP$'), ('definition', 'NN'), ('It', 'PRP'), ('is
', 'VBZ'), ('responsible', 'JJ'), ('for', 'IN'), ('text', 'JJ'), ('reading', 'NN'), ('in'
, 'IN'), ('a', 'DT'), ('language', 'NN'), ('It', 'PRP'), ('is', 'VBZ'), ('also', 'RB'), (
'called', 'VBN'), ('grammatical', 'JJ'), ('tagging', 'NN'), ('.', '.')]
```

```
#Code for Parsing
import nltk
```

```
from nltk import word_tokenize
from nltk.parse import RecursiveDescentParser
sentence = "the cat sat on the mat"
tokens = word_tokenize(sentence)
# Define a grammar for parsing the sentence
grammar = nltk.CFG.fromstring("""
            S -> NP VP
            NP -> DT NN
            VP -> V PP
            PP -> IN NP
            DT -> 'the'
            NN -> 'cat' | 'mat'
            V -> 'sat'
            IN -> 'on'
            """)
# Create a parser object
parser = RecursiveDescentParser(grammar)
print("\nAfter parsing: ")
for tree in parser.parse(tokens):
   print(tree)
```

Output



```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=================== RESTART: C:\Users\sumit\Desktop\Lab 9.py ===================
After parsing:
(S
  (NP (DT the) (NN cat))
  (VP (V sat) (PP (IN on) (NP (DT the) (NN mat)))))
```

EXPERIMENT-10

AIM: To implement 0/1 Knapsack using Brute Force approach

THEORY:
A set of objects with different weights and their values is given. The objective of 0/1 knapsack problem is to find the most optimal subset of these objects such that the total weight of these objects should not exceed a certain amount and the summation of values of these objects should be the maximum possible value incurred.

Brute Force approach is a method in which all 2^n combinations of the objects are taken 1 by 1 and the combination satisfying the above constraints is selected. E.g., if the objects with weights and values are given as follows and the maximum capacity is 40.

Item	Weight	Value
0	30	10
1	10	20
2	40	30
3	20	40

The most optimal solution is: Item1 and Item3 which account for a profit of 60 and a weight of 30.

CODE:
```
def knapsack_brute_force(values, weights, capacity):
    n = len(values)
    max_value = 0
    best_subset = None
    # Generate all possible subsets of items
    for i in range(2**n):
        subset = []
        subset_weight = 0
        subset_value = 0
        for j in range(n):
            if (i >> j) & 1==1:
                subset.append(j)
                subset_weight += weights[j]
                subset_value += values[j]
        # Check if the subset is feasible and has higher value than previous best
        if subset_weight <= capacity and subset_value > max_value:
            max_value = subset_value
            best_subset = subset
    return max_value, best_subset
knapsack_brute_force([10,20,30,40],[30,10,40,20],40)
```

OUTPUT:
```
(60, [1, 3])
```

EXPERIMENT-11

AIM: To study about Fuzzy Scikit Libraries

THEORY:

Fuzzy logic is a branch of mathematics that deals with reasoning that is approximate rather than fixed and exact. It is used in various applications, such as control systems, decision making, and data analysis, to deal with uncertainty and imprecision.

Scikit-fuzzy is a fuzzy logic toolkit for Python that provides a comprehensive set of tools for fuzzy control, fuzzy reasoning, and fuzzy analysis. It is built on top of the SciPy and NumPy libraries and provides a user-friendly interface for working with fuzzy logic.

In this experiment, we will use the scikit-fuzzy library to implement a fuzzy logic system to control the speed of a fan. We will use a simple set of rules to adjust the speed of the fan based on the temperature and humidity of the room.

CODE:

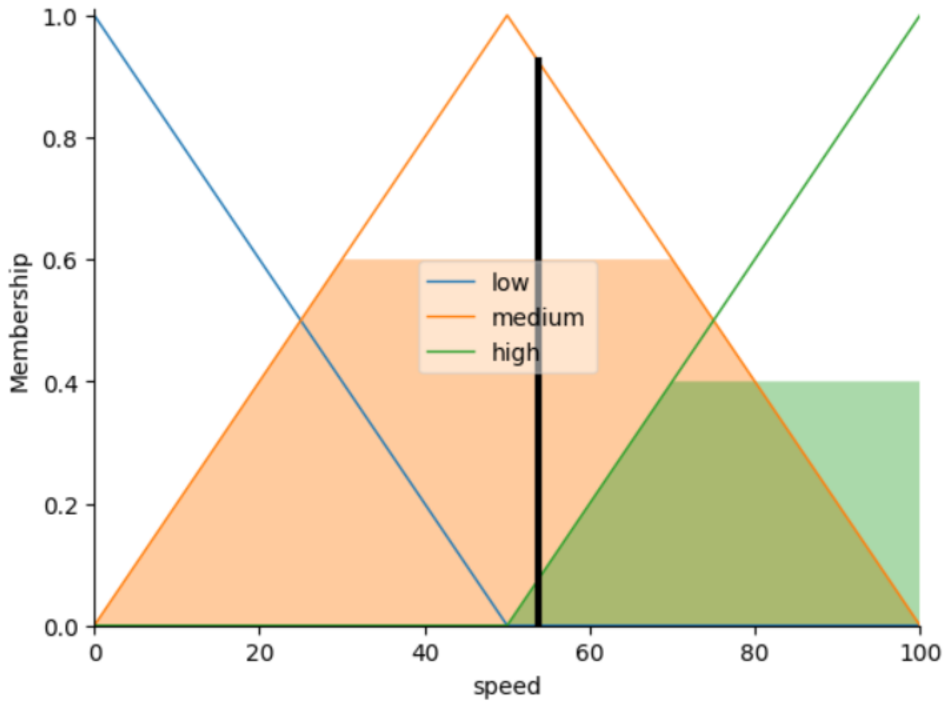
To get started, we need to install the scikit-fuzzy library. We can do this using pip:

```
!pip install scikit-fuzzy
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl

# Create input variables
temperature = ctrl.Antecedent(np.arange(0, 101, 1), 'temperature')
humidity = ctrl.Antecedent(np.arange(0, 101, 1), 'humidity')

# Create output variable
speed = ctrl.Consequent(np.arange(0, 101, 1), 'speed')

# Define fuzzy sets for input variables
temperature['cold'] = fuzz.trimf(temperature.universe, [0, 0, 50])
temperature['warm'] = fuzz.trimf(temperature.universe, [0, 50, 100])
temperature['hot'] = fuzz.trimf(temperature.universe, [50, 100, 100])

humidity['low'] = fuzz.trimf(humidity.universe, [0, 0, 50])
humidity['medium'] = fuzz.trimf(humidity.universe, [0, 50, 100])
humidity['high'] = fuzz.trimf(humidity.universe, [50, 100, 100])

# Define fuzzy sets for output variable
speed['low'] = fuzz.trimf(speed.universe, [0, 0, 50])
speed['medium'] = fuzz.trimf(speed.universe, [0, 50, 100])
speed['high'] = fuzz.trimf(speed.universe, [50, 100, 100])

# Define rules
rule1 = ctrl.Rule(temperature['cold'] & humidity['low'], speed['low'])
rule2 = ctrl.Rule(temperature['cold'] & humidity['medium'], speed['medium'])
```

```
rule3 = ctrl.Rule(temperature['cold'] & humidity['high'], speed['medium'])
rule4 = ctrl.Rule(temperature['warm'] & humidity['low'], speed['medium'])
rule5 = ctrl.Rule(temperature['warm'] & humidity['medium'], speed['medium'])
rule6 = ctrl.Rule(temperature['warm'] & humidity['high'], speed['high'])
rule7 = ctrl.Rule(temperature['hot'] & humidity['low'], speed['medium'])
rule8 = ctrl.Rule(temperature['hot'] & humidity['medium'], speed['high'])
rule9 = ctrl.Rule(temperature['hot'] & humidity['high'], speed['high'])

# Create control system
fan_ctrl = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5, rule6, rule7, rule8, rule9])
fan_speed = ctrl.ControlSystemSimulation(fan_ctrl)
# Set input values
fan_speed.input['temperature'] = 70
fan_speed.input['humidity'] = 60

# Compute result
fan_speed.compute()

# Print result
print(fan_speed.output['speed'])
speed.view(sim=fan_speed)
```

OUTPUT:

```
53.76811594202898
```


Membership
speed
low
medium
high