

DSA0314-Natural Language Processing for Programming Professionals

PROGRAMS FOR PRACTICAL WEEK

1. Write program demonstrates how to use regular expressions in Python to match and search for patterns in text.

PROGRAM:

```
import re

text1 = "DSA-0314 Natural Language Processing"
text2 = "DSA-0314 Natural Language"
word = "Natural Language Processing"
pattern = fr'\b{word}\b'

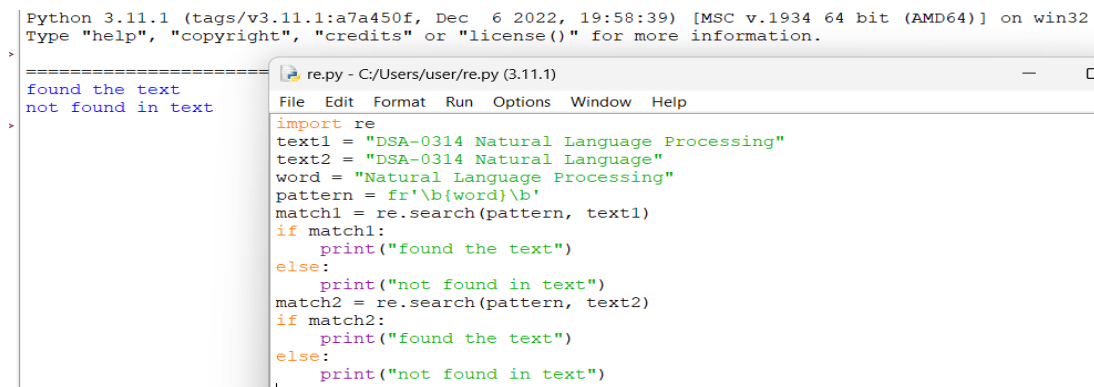
match1 = re.search(pattern, text1)

if match1:
    print("found the text")
else:
    print("not found in text")

match2 = re.search(pattern, text2)

if match2:
    print("found the text")
else:
    print("not found in text")
```

OUTPUT:



The screenshot displays a Python IDE window titled 're.py - C:/Users/user/re.py (3.11.1)'. The left pane shows the output of the program, which consists of two lines: 'found the text' and 'not found in text'. The right pane shows the source code of the program, which is the same code as provided in the 'PROGRAM' section. The code uses regular expressions to search for the word 'Natural Language Processing' in two different text strings. The first string contains the full phrase, so it is found. The second string only contains 'Natural Language', so it is not found.

```
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.
>
=====
found the text
not found in text
>

re.py - C:/Users/user/re.py (3.11.1)
File Edit Format Run Options Window Help
import re
text1 = "DSA-0314 Natural Language Processing"
text2 = "DSA-0314 Natural Language"
word = "Natural Language Processing"
pattern = fr'\b{word}\b'
match1 = re.search(pattern, text1)
if match1:
    print("found the text")
else:
    print("not found in text")
match2 = re.search(pattern, text2)
if match2:
    print("found the text")
else:
    print("not found in text")
```

2. Implement a basic finite state automaton that recognizes a specific language or pattern. In this example, we'll create a simple automaton to match strings ending with 'ab' using python.

PROGRAM:

```
INITIAL_STATE = 'q0'
ACCEPTING_STATE = 'q2'
TRANSITIONS = {
    'q0': {'a': 'q1', 'b': 'q0'},
    'q1': {'a': 'q1', 'b': 'q2'},
    'q2': {'a': 'q1', 'b': 'q0'}
}

def process_string(test_strings):
    current_state = INITIAL_STATE
    for char in test_strings:
        if char in TRANSITIONS.get(current_state, {}):
            current_state = TRANSITIONS[current_state][char]
        else:
            current_state = INITIAL_STATE
    return current_state == ACCEPTING_STATE

test_strings = ["ab", "aab", "bab", "bbaaab", "a", "b", "abc"]
for string in test_strings:
    if process_string(string):
        print(f'{string}' is accepted by the FSA.")
    else:
        print(f'{string}' is not accepted by the FSA.")
```

OUTPUT:

```
----- RESTART:
'ab' is accepted by the FSA.
'aab' is accepted by the FSA.
'bab' is accepted by the FSA.
'bbaaab' is accepted by the FSA.
'a' is not accepted by the FSA.
'b' is not accepted by the FSA.
'abc' is not accepted by the FSA.
|
```

3. Write program demonstrates how to perform morphological analysis using the NLTK library in Python.

PROGRAM:

```
import nltk

from nltk.tokenize import word_tokenize

nltk.download('punkt')

text = "Hello, Students Welcome to SSE."

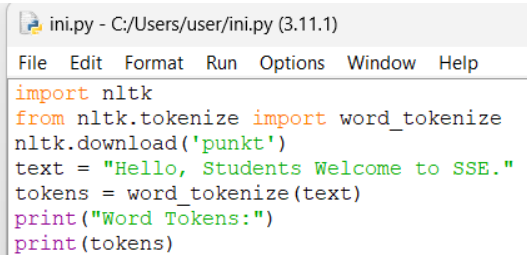
tokens = word_tokenize(text)

print("Word Tokens:")

print(tokens)
```

OUTPUT:

```
Word Tokens:
['Hello', ',', 'Students', 'Welcome', 'to', 'SSE', '.']
```

A screenshot of a Python IDE window titled 'ini.py - C:/Users/user/ini.py (3.11.1)'. The window contains the same Python code as shown in the 'PROGRAM' section. The code is color-coded: 'import' is orange, 'from' is orange, 'nltk' is blue, 'word_tokenize' is orange, 'download' is blue, 'text' is green, 'tokens' is blue, 'print' is green, and string literals are green. The output of the program is visible in the console area below the code editor, showing 'Word Tokens:' followed by a list of tokens: ['Hello', ',', 'Students', 'Welcome', 'to', 'SSE', '.'] on a new line.

```
ini.py - C:/Users/user/ini.py (3.11.1)
File Edit Format Run Options Window Help
import nltk
from nltk.tokenize import word_tokenize
nltk.download('punkt')
text = "Hello, Students Welcome to SSE."
tokens = word_tokenize(text)
print("Word Tokens:")
print(tokens)
```

4. Implement a finite-state machine for morphological parsing. In this example, we'll create a simple machine to generate plural forms of English nouns using python.

PROGRAM:

```
states = {

    "q0": {"other": "q0", "s": "q1"}

}

start_state = "q0"

final_states = {"q1"}


def process(word):

    current_state = start_state

    for char in word:

        if char == 's':

            current_state = states[current_state].get('s')
```

```

else:
    current_state = states[current_state].get('other')
    return current_state in final_states
words = ["cakes", "monkeys", "apple", "banana"]
for word in words:
    if process(word):
        print(f"The word '{word}' is a valid plural form.")
    else:
        print(f"The word '{word}' is not a valid plural form.")

```

OUTPUT:

```

The word 'cakes' is a valid plural form.
The word 'monkeys' is a valid plural form.
The word 'apple' is not a valid plural form.
The word 'banana' is not a valid plural form.

```

5. Use the Porter Stemmer algorithm to perform word stemming on a list of words using python libraries.

PROGRAM:

```

import nltk

from nltk.stem import PorterStemmer

nltk.download('punkt')

stemmer = PorterStemmer()

words = ["running", "jumping", "happiness", "computers", "generous"]

stemmed_words = [stemmer.stem(word) for word in words]

print(stemmed_words)

```

OUTPUT:

```

['run', 'jump', 'happi', 'comput', 'gener']

```

6. Implement a basic N-gram model for text generation. For example, generate text using a bigram model using python.

PROGRAM:

```

import nltk

nltk.download('treebank')

from nltk.corpus import treebank

from nltk.tag import BigramTagger

train_sents = treebank.tagged_sents()[[:3000]

bigram_tagger = BigramTagger(train_sents)

test_sents = treebank.sents()[3000:3010]

for sent in test_sents:

    tagged_sent = bigram_tagger.tag(sent)

    print(tagged_sent)

```

OUTPUT:

```

[nltk_data] Downloading package treebank to /root/nltk_data...
[nltk_data]   Unzipping corpora/treebank.zip.
[('At', 'IN'), ('Tokyo', 'NNP'), (',', ','), ('the', 'DT'), ('Nikkei', None), ('index', None), ('of',
[('In', 'IN'), ('early', 'JJ'), ('trading', 'NN'), ('in', 'IN'), ('Tokyo', 'NNP'), ('Thursday', 'NNP
[('Wednesday', 'NNP'), ('"', 'POS'), ('volume', None), ('on', None), ('the', None), ('First', None)

```

7. Write program using the NLTK library to perform part-of-speech tagging on a text.

PROGRAM:

```

import nltk

def pos_tagging(text):

    words = nltk.word_tokenize(text)

    pos_tags = nltk.pos_tag(words)

    return pos_tags

text = "Good Morning students"

text1="Pay attention please"

text2="He is the founder of our university"

tags = pos_tagging(text)

print(tags)

```

```

tagss = pos_tagging(text1)

print(tagss)

tagsss = pos_tagging(text2)

print(tagsss)

```

OUTPUT:

```

ini.py - C:/Users/user/ini.py (3.11.1)
File Edit Format Run Options Window Help
import nltk
def pos_tagging(text):
    words = nltk.word_tokenize(text)
    pos_tags = nltk.pos_tag(words)
    return pos_tags
text = "Good Morning students"
text1="Pay attention please"
text2="He is the founder of our university"
tags = pos_tagging(text)
print(tags)
tagss = pos_tagging(text1)
print(tagss)
tagsss = pos_tagging(text2)
print(tagsss)

Ln: 13 Col: 27 ART: C:/Users/user/ini.py =====
Generated text: dog . the quick brown fox jumps over the quick brown fox jumps over the
===== RESTART: C:/Users/user/ini.py =====
[('Good', 'JJ'), ('Morning', 'NN'), ('students', 'NNS')]
[('Pay', 'NNP'), ('attention', 'NN'), ('please', 'NN')]
[('He', 'PRP'), ('is', 'VBZ'), ('the', 'DT'), ('founder', 'NN'), ('of', 'IN'), ('our', 'PRP$'), ('university', 'NN')]

```

- Implement a simple stochastic part-of-speech tagging algorithm using a basic probabilistic model to assign POS tags using python.

PROGRAM:

```

corpus = [
    (["the", "cat", "sat"], ["DET", "NOUN", "VERB"]),
    (["the", "dog", "barked"], ["DET", "NOUN", "VERB"]),
    (["a", "cat", "meowed"], ["DET", "NOUN", "VERB"]),
    (["the", "dog", "ran"], ["DET", "NOUN", "VERB"])
]

```

```
word_tag_probs = {}
```

```
for sentence, tags in corpus:
```

```
    for word, tag in zip(sentence, tags):
```

```
        if word not in word_tag_probs:
```

```
            word_tag_probs[word] = {}
```

```
        if tag not in word_tag_probs[word]:
```

```

        word_tag_probs[word][tag] = 0
    word_tag_probs[word][tag] += 1
for word in word_tag_probs:
    total = sum(word_tag_probs[word].values())
    for tag in word_tag_probs[word]:
        word_tag_probs[word][tag] /= total
def simple_pos_tag(sentence):
    tags = []
    for word in sentence:
        if word in word_tag_probs:
            tag = max(word_tag_probs[word], key=word_tag_probs[word].get)
        else:
            tag = "NOUN"
        tags.append(tag)
    return tags
new_sentence = ["the", "cat", "ran"]
tags = simple_pos_tag(new_sentence)
print(list(zip(new_sentence, tags)))

```

OUTPUT:

```

=====
[('the', 'DET'), ('cat', 'NOUN'), ('ran', 'VERB')]
|

```

9. Implement a rule-based part-of-speech tagging system using regular expressions using python.

PROGRAM:

```

import re
rules = [
    (r'\bthe\b', 'DET'),
    (r'\ba\b', 'DET'),
    (r'\ban\b', 'DET'),

```

```

(r'\b(cat|dog)\b', 'NOUN'),
(r'\b(sat|barked|meowed|ran)\b', 'VERB'),
(r'\b\w+ly\b', 'ADV'),
(r'\b\w+ing\b', 'VERB'),
(r'\b\w+ed\b', 'VERB'),
(r'\b\w+s\b', 'NOUN'),
]

```

```
def rule_based_pos_tag(sentence):
```

```
    tags = []
```

```
    words = sentence.split()
```

```
    for word in words:
```

```
        tag = 'NOUN'
```

```
        for pattern, rule_tag in rules:
```

```
            if re.fullmatch(pattern, word):
```

```
                tag = rule_tag
```

```
                break
```

```
        tags.append((word, tag))
```

```
    return tags
```

```
sentence = "the cat sat on the mat and the dog barked"
```

```
tagged_sentence = rule_based_pos_tag(sentence)
```

```
print(tagged_sentence)
```

OUTPUT:

```

====
[('the', 'DET'), ('cat', 'NOUN'), ('sat', 'VERB'), ('on', 'NOUN'), ('the', 'DET'), ('mat', 'NOUN'), ('and', 'NOUN'), ('the', 'DET'), ('dog', 'NOUN'), ('barked', 'VERB')]
|

```

10. Implement transformation-based tagging using a set of transformation rules, apply a simple rule to tag words using python.

PROGRAM:

```
corpus = [
```



```

["I", "want", "to", "book", "a", "flight"],
["the", "book", "is", "on", "the", "table"]
]

tagged_corpus = [[(word, "NOUN") for word in sentence] for sentence in corpus]

rules = [
    ("NOUN", "VERB", lambda word, prev: word == "book" and prev == "to"),
]

def apply_rules(tagged_sentence, rules):
    return [(word, next((new_tag for current_tag, new_tag, condition in rules if tag == current_tag
and condition(word, tagged_sentence[i-1][0] if i > 0 else None)), tag)) for i, (word, tag) in
enumerate(tagged_sentence)]

transformed_corpus = [apply_rules(sentence, rules) for sentence in tagged_corpus]

for sentence in transformed_corpus:
    print(sentence)

```

OUTPUT:

```

[('I', 'NOUN'), ('want', 'NOUN'), ('to', 'NOUN'), ('book', 'VERB'), ('a', 'NOUN'), ('flight', 'NOUN')]
[('the', 'NOUN'), ('book', 'NOUN'), ('is', 'NOUN'), ('on', 'NOUN'), ('the', 'NOUN'), ('table', 'NOUN')]
|

```

11. Implement a simple top-down parser for context-free grammars using python.

PROGRAM:

```

import nltk

from nltk import CFG, RecursiveDescentParser

grammar = CFG.fromstring("""
    S -> NP VP
    NP -> Det N | Det N PP | 'I'
    VP -> V NP | VP PP
    PP -> P NP
    Det -> 'a' | 'the'
    N -> 'man' | 'park' | 'dog' | 'telescope'
    V -> 'saw' | 'walked'
    P -> 'in' | 'with'

```

```
parser = RecursiveDescentParser(grammar)
sentence = "I saw the man in the park".split()
for tree in parser.parse(sentence):
    print(tree)
    tree.pretty_print()
```

```
(S
  (NP I)
  (VP
    (V saw)
    (NP (Det the) (N man) (PP (P in) (NP (Det the) (N park))))
  )
)

      |
      |-----VP
      |
      |-----NP
      |
      |-----PP
      |
      |-----NP
      |
      |-----N
      |
      |
NP  V  Det  N  P  Det  N
|   |   |   |   |   |
I  saw the man in the      park

(S
  (NP I)
  (VP
    (VP (V saw) (NP (Det the) (N man)))
    (PP (P in) (NP (Det the) (N park))))
  )
)

      |
      |-----VP
      |
      |-----VP
      |
      |-----NP
      |
      |-----PP
      |
      |-----NP
      |
      |-----N
      |
      |
NP  V  Det  N  P  Det  N
|   |   |   |   |   |
I  saw the      man in the      park
```

PROGRAM:

```
class EarleyParser:

    def __init__(self, grammar):

        self.grammar = grammar

        self.chart = []

    def parse(self, tokens):

        self.chart = [[] for _ in range(len(tokens) + 1)]

        start_rule = self.grammar[0]

        start_state = (start_rule[0], [], start_rule[1], 0)

        self.chart[0].append(start_state)

        for i in range(len(tokens) + 1):

            for state in self.chart[i]:
```

```

        if self.is_complete(state):
            self.completer(state, i)
        elif self.is_next_non_terminal(state):
            self.predictor(state, i)
        else:
            if i < len(tokens):
                self.scanner(state, i, tokens[i])

    for state in self.chart[-1]:
        if state[0] == start_rule[0] and self.is_complete(state) and state[3] == 0:
            return True
    return False

def is_complete(self, state):
    return len(state[1]) == len(state[2])

def is_next_non_terminal(self, state):
    return len(state[1]) < len(state[2]) and state[2][len(state[1])].isupper()

def predictor(self, state, index):
    next_non_terminal = state[2][len(state[1])]
    for rule in self.grammar:
        if rule[0] == next_non_terminal:
            new_state = (rule[0], [], rule[1], index)
            if new_state not in self.chart[index]:
                self.chart[index].append(new_state)

def scanner(self, state, index, token):
    next_symbol = state[2][len(state[1])]
    if next_symbol == token:
        new_state = (state[0], state[1] + [token], state[2], state[3])
        if new_state not in self.chart[index + 1]:
            self.chart[index + 1].append(new_state)

def completer(self, state, index):

```

```

for old_state in self.chart[state[3]]:
    if len(old_state[1]) < len(old_state[2]) and old_state[2][len(old_state[1])] == state[0]:
        new_state = (old_state[0], old_state[1] + [state[0]], old_state[2], old_state[3])
        if new_state not in self.chart[index]:
            self.chart[index].append(new_state)
if __name__ == "__main__":
    grammar = [
        ('S', ['NP', 'VP']),
        ('NP', ['Det', 'N']),
        ('VP', ['V', 'NP']),
        ('Det', ['the']),
        ('N', ['dog']),
        ('V', ['chases']),
    ]
    parser = EarleyParser(grammar)
    tokens = ['the', 'dog', 'chases', 'the', 'dog']
    print(parser.parse(tokens))

```

OUTPUT:

```
| False
```

13. Generate a parse tree for a given sentence using a context-free grammar using python program.

PROGRAM:

```

import nltk

from nltk import CFG, RecursiveDescentParser

grammar = CFG.fromstring("""
    S -> NP VP
    NP -> Det N | Det N PP | 'I'
    VP -> V NP | VP PP
    PP -> P NP
    Det -> 'a' | 'the'

```

N -> 'man' | 'park' | 'dog' | 'telescope'

V -> 'saw' | 'walked'

P -> 'in' | 'with'

"""

parser = RecursiveDescentParser(grammar)

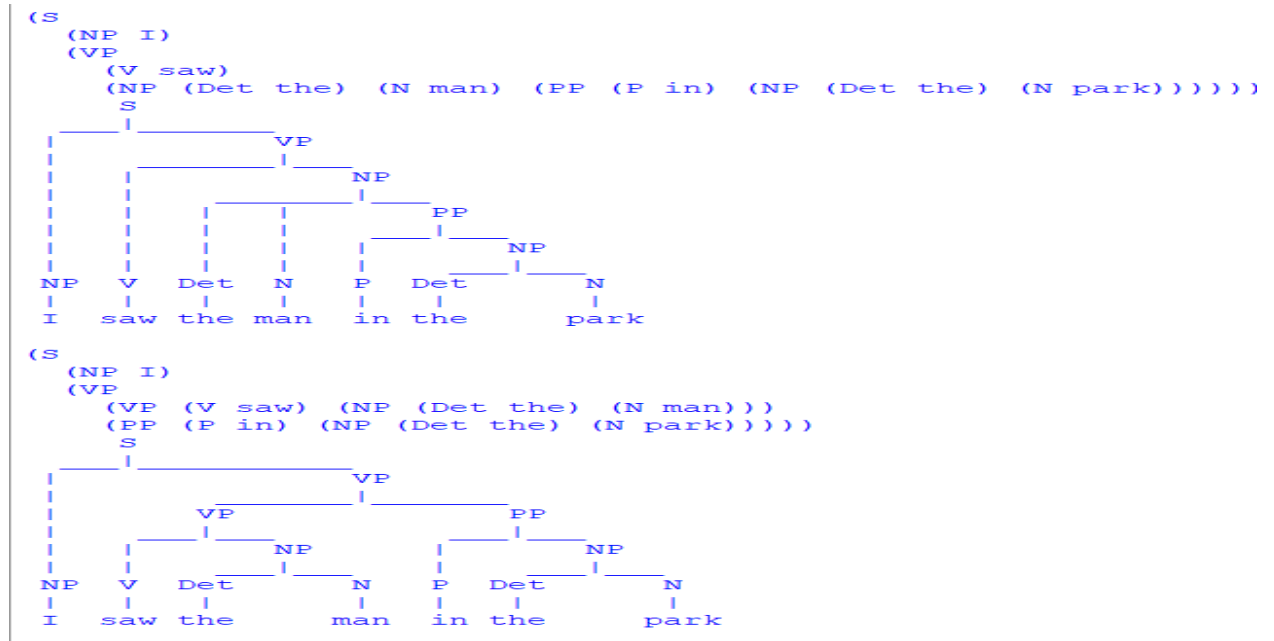
sentence = "I saw the man in the park".split()

for tree in parser.parse(sentence):

print(tree)

tree.pretty_print()

OUTPUT:



14. Create a program in python to check for agreement in sentences based on a context-free grammar's rules.

PROGRAM:

import nltk

from nltk import CFG

grammar = CFG.fromstring("""

S -> NP_SG VP_SG | NP_PL VP_PL

NP_SG -> Det_SG N_SG

```

NP_PL -> Det_PL N_PL
VP_SG -> V_SG
VP_PL -> V_PL
Det_SG -> 'the'
Det_PL -> 'the'
N_SG -> 'cat' | 'dog'
N_PL -> 'cats' | 'dogs'
V_SG -> 'runs' | 'jumps'
V_PL -> 'run' | 'jump'
"""
)

```

```

def check_agreement(sentence):
    tokens = sentence.split()
    parser = nltk.ChartParser(grammar)

    try:
        next(parser.parse(tokens))
        return True

    except StopIteration:
        return False

```

```

sentences = [
    "the cat runs",
    "the dogs run",
    "the cat run",
    "the dog runs"
]

```

```

for sentence in sentences:

```

```

if check_agreement(sentence):

    print(f"Agreement satisfied for '{sentence}'")

else:

    print(f"Agreement NOT satisfied for '{sentence}'")

```

OUTPUT:

```

-----
Agreement satisfied for 'the cat runs'
Agreement satisfied for 'the dogs run'
Agreement NOT satisfied for 'the cat run'
Agreement satisfied for 'the dog runs'

```

15. Implement probabilistic context-free grammar parsing for a sentence using python.

PROGRAM:

```

import nltk

from nltk import PCFG, ViterbiParser

```

```

grammar = PCFG.fromstring("""
    S -> NP VP [1.0]
    VP -> V NP [0.7] | V [0.3]
    NP -> Det N [0.6] | N [0.4]
    Det -> 'the' [0.8] | 'a' [0.2]
    N -> 'cat' [0.5] | 'dog' [0.5]
    V -> 'chased' [0.9] | 'saw' [0.1]
""")

```

```

def parse_sentence_pcfg(sentence):

    tokens = sentence.split()

    parser = ViterbiParser(grammar)

    try:

        trees = list(parser.parse(tokens))

```

```

        return trees[0]

    except IndexError:

        return None # Handle case where no parse tree is found

sentences = [

    "the cat chased the dog",

    "a dog saw the cat"

]

for sentence in sentences:

    parse_tree = parse_sentence_pcfg(sentence)

    if parse_tree:

        print(f"Parse tree for '{sentence}':")

        print(parse_tree)

        parse_tree.pretty_print()

    else:

        print(f"No parse tree found for sentence: '{sentence}'")

```

OUTPUT:

```

Parse tree for 'the cat chased the dog':
(S
  (NP (Det the) (N cat))
  (VP (V chased) (NP (Det the) (N dog)))) (p=0.036288)

```

```

      S
     / \
    NP  VP
   / \  / \
  Det N V  NP
  |  |  | / \
  |  |  | Det N
  |  |  | |  |
the cat chased the dog

```

```

Parse tree for 'a dog saw the cat':
(S
  (NP (Det a) (N dog))
  (VP (V saw) (NP (Det the) (N cat)))) (p=0.001008)

```

```

      S
     / \
    NP  VP
   / \  / \
  Det N V  NP
  |  |  | / \
  |  |  | Det N
  |  |  | |  |
a   dog saw the cat

```

16. Implement a Python program using the SpaCy library to perform Named Entity Recognition (NER) on a given text.

PROGRAM:


```

import spacy

nlp = spacy.load('en_core_web_sm')

text = "Apple is looking at buying U.K. startup for $1 billion"

doc = nlp(text)

for token in doc:

    print(token.text, token.pos_)

```

OUTPUT:

```

Apple PROP
is AUX
looking VERB
at ADP
buying VERB
U.K. PROP
startup NOUN
for ADP
$ SYM
1 NUM
billion NUM

```

17. Write program demonstrates how to access WordNet, a lexical database, to retrieve synsets and explore word meanings in python.

PROGRAM:

```

import nltk

from nltk.corpus import wordnet

def explore_word_meanings(word):

    synsets = wordnet.synsets(word)

    if not synsets:

        print(f"No meanings found for '{word}' in WordNet.")

        return

    for synset in synsets:

        print(f"Meaning ({synset.pos()}) - {synset.definition()}")

        print(f"Example: {synset.examples()}")

        print()

word_to_explore = "bank"

explore_word_meanings(word_to_explore)

```

OUTPUT:

```

Meaning (n) - sloping land (especially the slope beside a body of water)
Example: ['they pulled the canoe up on the bank', 'he sat on the bank of the river and watched the currents']

Meaning (n) - a financial institution that accepts deposits and channels the money into lending activities
Example: ['he cashed a check at the bank', 'that bank holds the mortgage on my home']

Meaning (n) - a long ridge or pile
Example: ['a huge bank of earth']

Meaning (n) - an arrangement of similar objects in a row or in tiers
Example: ['he operated a bank of switches']

Meaning (n) - a supply or stock held in reserve for future use (especially in emergencies)
Example: []

Meaning (n) - the funds held by a gambling house or the dealer in some gambling games
Example: ['he tried to break the bank at Monte Carlo']

```

18. Implement a simple FOPC parser for basic logical expressions using python program.

PROGRAM:

18th code First Order Predicate Logic:

!pip install transformers

!pip install torch

import nltk

from nltk import CFG

fopc_grammar = CFG.fromstring("""

S -> EXPR

EXPR -> PRED | EXPR 'AND' EXPR | EXPR 'OR' EXPR

PRED -> NAME '(' VARS ')'

VARS -> VAR | VAR ',' VARS

NAME -> 'P' | 'Q' | 'R'

VAR -> 'x' | 'y' | 'z'

AND -> 'AND'

OR -> 'OR'

""")

parser = nltk.ChartParser(fopc_grammar)

def parse_expression(expression):

try:

tokens = expression.replace('(', ' (').replace(')', ') ').replace(',', ' , ').split()

trees = list(parser.parse(tokens))

if trees:

for tree in trees:

tree.pretty_print()

```

else:
    print("No valid parse found.")

except ValueError as e:
    print(f"Error: {e}")

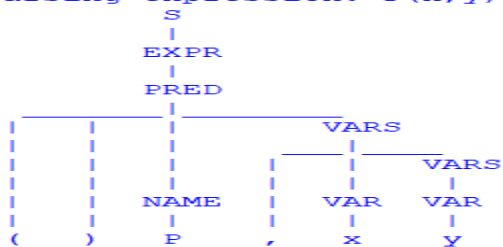
expressions = [
    "P(x,y)",
    "Q(x) AND R(y,z)",
    "P(x) OR Q(y) AND R(z)"
]

for expr in expressions:
    print(f"Parsing expression: {expr}")
    parse_expression(expr)
    print()

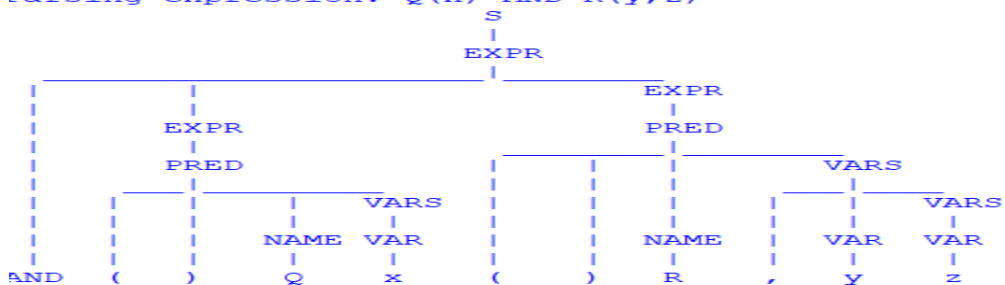
```

OUTPUT:

Parsing expression: P (x,y)



Parsing expression: Q (x) AND R (y, z)



19. Create a program for word sense disambiguation using the Lesk algorithm using python.

PROGRAM:

```

from nltk.corpus import stopwords, wordnet
from nltk.tokenize import word_tokenize

```

```

from nltk.wsd import lesk

import string

def preprocess_text(text):

    tokens = word_tokenize(text.lower())

    tokens = [token for token in tokens if token not in string.punctuation and token not in
stopwords.words('english')]

    return tokens

def perform_wsd(word, sentence):

    best_sense = lesk(sentence, word)

    return best_sense

sentence = "He went to the bank to deposit his money."
word = "bank"

preprocessed_sentence = preprocess_text(sentence)

sense = perform_wsd(word, preprocessed_sentence)

if sense:

    print(f"Word: {word}")

    print(f"Sentence: {sentence}")

    print(f"Best Sense: {sense.name()} - {sense.definition()}")

else:

    print(f"No suitable sense found for '{word}' in the context.")

```

OUTPUT:

```

Word: bank
Sentence: He went to the bank to deposit his money.
Best Sense: savings_bank.n.02 - a container (usually with a slot in the top) for keeping money at home

```

20. Implement a basic information retrieval system using TF-IDF (Term Frequency-Inverse Document Frequency) for document ranking using python.

PROGRAM:

```

import math

from nltk.corpus import stopwords

from nltk.tokenize import word_tokenize

```

```

from nltk.probability import FreqDist

import string

documents = {
    'doc1': "TF-IDF is a technique used in information retrieval and text mining.",
    'doc2': "The TF-IDF score is used to determine the importance of a term within a document.",
    'doc3': "Information retrieval is the process of obtaining information from a collection of text documents."
}

def preprocess_text(text):
    tokens = word_tokenize(text.lower())

    tokens = [token for token in tokens if token not in string.punctuation and token not in stopwords.words('english')]

    return tokens

def compute_tf(document):
    fd = FreqDist(document)

    tf = {word: fd[word] / len(document) for word in fd}

    return tf

def compute_idf(corpus):
    all_words = set(word for doc in corpus for word in doc)

    idf = {word: math.log(len(corpus) / (1 + sum(1 for doc in corpus if word in doc))) for word in all_words}

    return idf

def compute_tfidf(documents, idf):
    tfidf_scores = {}

    for doc_id, doc_text in documents.items():
        tokens = preprocess_text(doc_text)

        tf = compute_tf(tokens)

        tfidf_scores[doc_id] = sum(tf[word] * idf.get(word, 0) for word in tf)

    return tfidf_scores

```

```

def rank_documents(query, documents):
    preprocessed_query = preprocess_text(query)
    idf = compute_idf([preprocessed_query] + [preprocess_text(doc) for doc in documents.values()])
    tfidf_scores = compute_tfidf(documents, idf)
    ranked_documents = sorted(tfidf_scores.items(), key=lambda x: x[1], reverse=True)
    return ranked_documents

query = "TF-IDF information retrieval"
ranked_docs = rank_documents(query, documents)
print(f"Query: {query}")

for idx, (doc_id, score) in enumerate(ranked_docs, start=1):
    print(f"Rank {idx}: Document '{doc_id}' with TF-IDF Score {score:.4f}")

```

OUTPUT:

```

Query: TF-IDF information retrieval
Rank 1: Document 'doc2' with TF-IDF Score 0.5558
Rank 2: Document 'doc3' with TF-IDF Score 0.3825
Rank 3: Document 'doc1' with TF-IDF Score 0.2802

```

21. Create a python program that performs syntax-driven semantic analysis by extracting noun phrases and their meanings from a sentence.

PROGRAM:

```

import nltk

from nltk.corpus import wordnet as wn

nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
nltk.download('punkt')

def get_noun_phrases(sentence):
    words = nltk.word_tokenize(sentence)
    pos_tags = nltk.pos_tag(words)
    noun_phrases = []

    for word, pos in pos_tags:
        if pos.startswith('NN'):
            noun_phrases.append(word)

    return noun_phrases

```

```

def get_meaning(word):

    synsets = wn.synsets(word, pos=wn.NOUN)

    if synsets:

        return synsets[0].definition()

    return None

sentence = "The quick brown fox jumps over the lazy dog."

noun_phrases = get_noun_phrases(sentence)

for noun in noun_phrases:

    meaning = get_meaning(noun)

    if meaning:

        print(f"Noun: {noun} -> Meaning: {meaning}")

```

OUTPUT:

```

Noun: brown -> Meaning: an orange of low brightness and saturation
Noun: fox -> Meaning: alert carnivorous mammal with pointed muzzle and ears and a bushy tail; most are predators that do not hunt in packs
Noun: dog -> Meaning: a member of the genus Canis (probably descended from the common wolf) that has been domesticated by man since prehistoric times; occurs in many breeds

```

22. Create a python program that performs reference resolution within a text.

PROGRAM:

```

from nltk.tokenize import word_tokenize, sent_tokenize

from nltk.tag import pos_tag

text = "John went to the store. He bought some milk."

sentences = sent_tokenize(text)

tokenized_sentences = [word_tokenize(sentence) for sentence in sentences]

tagged_sentences = [pos_tag(sentence) for sentence in tokenized_sentences]

def resolve_references(tagged_sentences):

    resolved_text = []

    prev_noun = None

    for sentence in tagged_sentences:

        new_sentence = []

        for word, tag in sentence:

            if tag in ['NN', 'NNP'] and prev_noun is None:

                prev_noun = word

```

```

        if word.lower() == 'he' and prev_noun:
            new_sentence.append(prev_noun)
        else:
            new_sentence.append(word)
        resolved_text.append(" ".join(new_sentence))
    return " ".join(resolved_text)

resolved_text = resolve_references(tagged_sentences)

print("Original Text: ", text)
print("Resolved Text: ", resolved_text)

```

OUTPUT:

```

Original Text:  John went to the store. He bought some milk.
Resolved Text:  John went to the store . John bought some milk .

```

23. Develop a python program that evaluates the coherence of a given text.

PROGRAM:

```

import nltk

from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.corpus import stopwords

import string

nltk.download('punkt')
nltk.download('stopwords')

def preprocess_text(text):
    stop_words = set(stopwords.words('english') + list(string.punctuation))

    tokens = word_tokenize(text.lower())

    return [word for word in tokens if word not in stop_words]

def lexical_cohesion_score(tokens):
    score = 0

    seen_tokens = set()

    for token in tokens:
        if token in seen_tokens:

```



```

        score += 1

    seen_tokens.add(token)

    return score / len(tokens) if tokens else 0

text = "John went to the store. He bought some milk. The store was busy."
tokens = preprocess_text(text)
cohesion_score = lexical_cohesion_score(tokens)
print("Text: ", text)
print("Lexical Cohesion Score: ", cohesion_score)

```

OUTPUT:

```

Text:  John went to the store. He bought some milk. The store was busy.
Lexical Cohesion Score:  0.14285714285714285

```

24. Create a python program that recognizes dialog acts in a given dialog or conversation.

PROGRAM:

```

import nltk

from nltk.tokenize import word_tokenize

dialog = ["Hi, how are you?", "I'm good, thank you!", "What's your name?", "My name is John."]

def recognize_dialog_act(utterance):

    tokens = word_tokenize(utterance.lower())

    if tokens[0] in ['hi', 'hello']:

        return 'Greeting'

    elif tokens[-1] == '?':

        return 'Question'

    elif tokens[0] in ['thanks', 'thank']:

        return 'Thanks'

    else:

        return 'Statement'

for utterance in dialog:

    act = recognize_dialog_act(utterance)

    print(f"Utterance: '{utterance}' => Dialog Act: {act}")

```

OUTPUT:

```
Utterance: 'Hi, how are you?' => Dialog Act: Greeting
Utterance: 'I'm good, thank you!' => Dialog Act: Statement
Utterance: 'What's your name?' => Dialog Act: Question
Utterance: 'My name is John.' => Dialog Act: Statement
```

25. Utilize the GPT-3 model to generate text based on a given prompt. Make sure to install the OpenAI GPT-3 library in python implementation.

PROGRAM:

text generated prompt :

```
from transformers import pipeline
```

```
generator = pipeline('text-generation', model='gpt2', framework='pt')
```

```
def generate_text(prompt, max_length=100):
```

```
    response = generator(prompt, max_length=max_length, num_return_sequences=1)
```

```
    return response[0]['generated_text']
```

```
prompt = "Do you know India"
```

```
generated_text = generate_text(prompt)
```

```
print(f"Generated Text: {generated_text}")
```

OUTPUT:

```
Generated Text: Do you know India now?" and "When will we get here?"

I don't understand this question as much as this one. To start, I don't understand it. If India is not one, why would India be one? It is India.
he rest of America, including the U.S.A., I can see, like India, all over the planet and the rest of the world. I have the knowledge that these
```

26. Implement a machine translation program using the Hugging Face Transformers library, translate English text to French using python.

PROGRAM:

```
from transformers import MarianMTModel, MarianTokenizer
```

```
model_name = 'Helsinki-NLP/opus-mt-en-fr'
```

```
tokenizer = MarianTokenizer.from_pretrained(model_name)
```

```
model = MarianMTModel.from_pretrained(model_name)
```

```
def translate(text, src_lang="en", tgt_lang="fr"):
```

```
    translated = model.generate(**tokenizer(text, return_tensors="pt", padding=True))
```

```
    translated_text = [tokenizer.decode(t, skip_special_tokens=True) for t in translated]
```

```
    return translated_text[0]
```

```
if __name__ == "__main__":
```

```
english_text = "Hello, how are you?"  
french_translation = translate(english_text)  
print(f"English: {english_text}")  
print(f"French: {french_translation}")
```

OUTPUT:

```
English: Hello, how are you?  
French: Bonjour, comment allez-vous?
```