

## **DSA0311-NATURAL LANGUAGE PROCESSING LAB PROGRAMS**

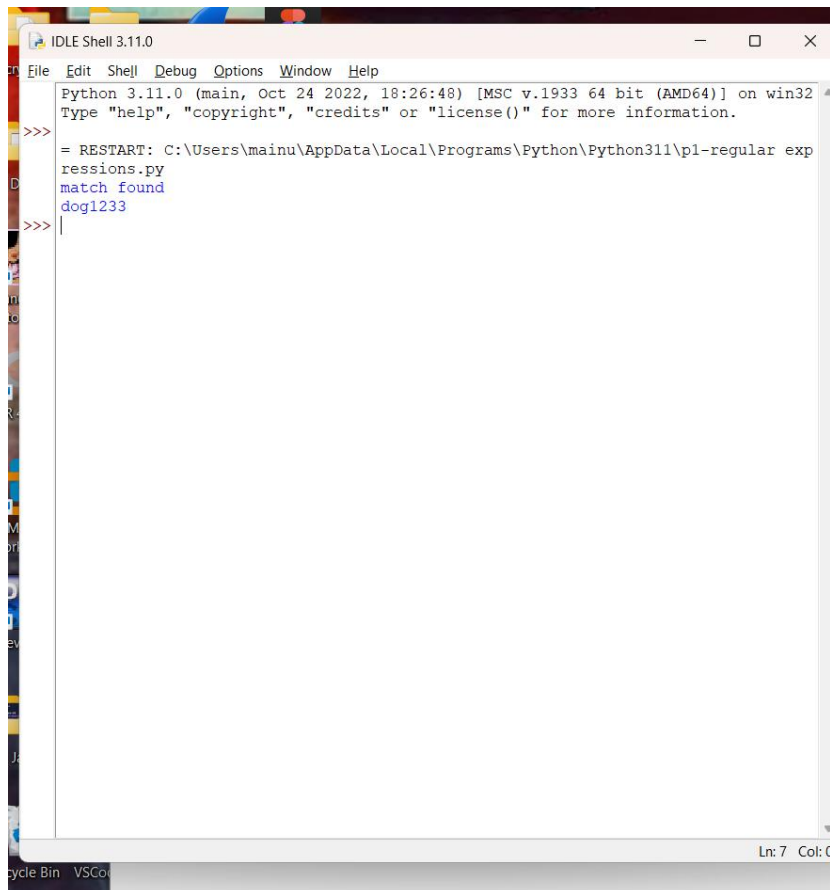
**192124049**

**SK.KHAJA MAINUDDIN**

Program-1:

```
import re
text="new dog1233"
word=r'\bdog[0-9]*\b'
match1=re.findall(word,text)
if match1:
    print("match found")
    for a in match1:
        print(a)
```

output:



```
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\mainu\AppData\Local\Programs\Python\Python311\pl-regular exp
ressions.py
match found
dog1233
>>>
|
```

program-2:

```
strings_to_test = ["ab", "abc", "defab", "abab",  
"ababab","abab"]
```

```
for string in strings_to_test:
```

```
    state = 0
```

```
    for char in string:
```

```
        if state == 0 and char == 'a':
```

```
            state = 1
```

```
        elif state == 1 and char == 'b':
```

```
            state = 2
```

```
else:

    state = 0

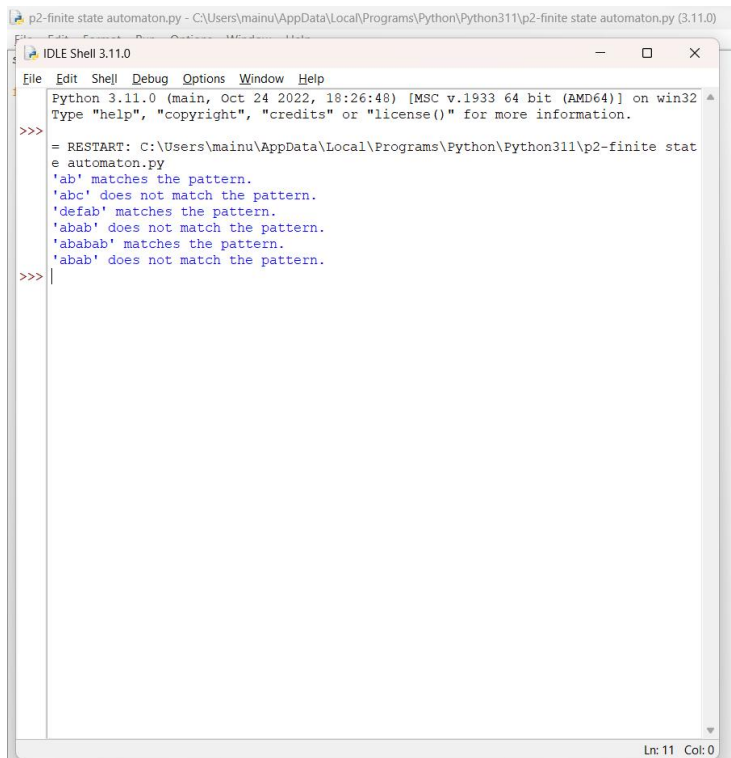
if state == 2:

    print(f'"{string}" matches the pattern.')

else:

    print(f'"{string}" does not match the pattern.')
```

output:



```
p2-finite state automaton.py - C:\Users\mainu\AppData\Local\Programs\Python\Python311\p2-finite state automaton.py (3.11.0)

IDLE Shell 3.11.0
File Edit Shell Debug Options Window Help
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\mainu\AppData\Local\Programs\Python\Python311\p2-finite state automaton.py
'ab' matches the pattern.
'abc' does not match the pattern.
'defab' matches the pattern.
'abab' does not match the pattern.
'ababab' matches the pattern.
'abab' does not match the pattern.
>>>
```

program-3:

```
from nltk import PorterStemmer

from nltk.tokenize import word_tokenize

import nltk
```

```
# Download necessary NLTK resources
```

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

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

```
nltk.download('words')
```

```
nltk.download('stopwords')
```

```
nltk.download('maxent_ne_chunker')
```

```
text = "you must be takecare while crossing road and also  
lookup the magically"
```

```
port = PorterStemmer()
```

```
words = word_tokenize(text)
```

```
print(words)
```

```
stem1 = [port.stem(word) for word in words]
```

```
print(stem1)
```

output:

```
IDLE Shell 3.11.0
File Edit Shell Debug Options Window Help
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\mainu\AppData\Local\Programs\Python\Python311\p3-morphological using NLTK.py
[nltk_data] Downloading package punkt to
[nltk_data] C:\Users\mainu\AppData\Roaming\nltk_data...
[nltk_data] Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data] C:\Users\mainu\AppData\Roaming\nltk_data...
[nltk_data] Package wordnet is already up-to-date!
[nltk_data] Downloading package words to
[nltk_data] C:\Users\mainu\AppData\Roaming\nltk_data...
[nltk_data] Package words is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data] C:\Users\mainu\AppData\Roaming\nltk_data...
[nltk_data] Package stopwords is already up-to-date!
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data] C:\Users\mainu\AppData\Roaming\nltk_data...
[nltk_data] Package maxent_ne_chunker is already up-to-date!
['you', 'must', 'be', 'takecare', 'while', 'crossing', 'road', 'and', 'also', 'lookup', 'the', 'magically']
['you', 'must', 'be', 'takecare', 'while', 'cross', 'road', 'and', 'also', 'lookup', 'the', 'magic']
>>>
```

program-4:

```
irregular_nouns = {'man': 'men', 'woman': 'women', 'child':  
'children'}
```

```
words_to_pluralize = ['cat', 'dog', 'man', 'woman', 'bus', 'box',  
'church']
```

```
for word in words_to_pluralize:
```

```
    if word in irregular_nouns:
```

```
        plural_form = irregular_nouns[word]
```

```
        elif word.endswith('s') or word.endswith('x') or  
word.endswith('z'):
```

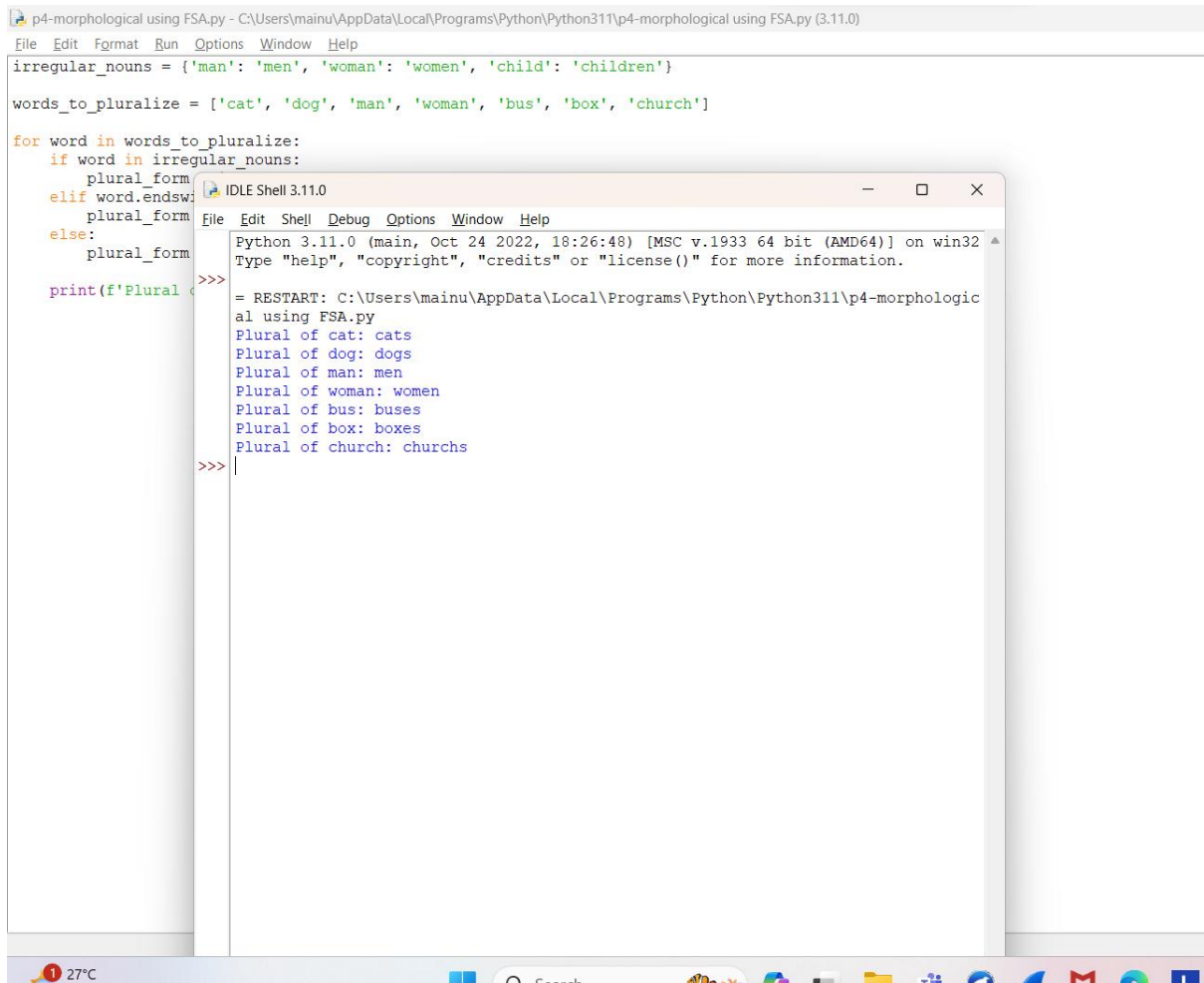
```
            plural_form = word + 'es'
```

```
    else:
```

```
plural_form = word + 's'
```

```
print(f'Plural of {word}: {plural_form}')
```

output:



The screenshot shows a Python IDE window titled 'p4-morphological using FSA.py'. The script defines a dictionary of irregular nouns and a list of words to pluralize. It then iterates through the words, checking if they are in the irregular nouns dictionary. If not, it appends 's' to the word. The output is printed as 'Plural of {word}: {plural\_form}'.

```
p4-morphological using FSA.py - C:\Users\mainu\AppData\Local\Programs\Python\Python311\p4-morphological using FSA.py (3.11.0)
File Edit Format Run Options Window Help
irregular_nouns = {'man': 'men', 'woman': 'women', 'child': 'children'}
words_to_pluralize = ['cat', 'dog', 'man', 'woman', 'bus', 'box', 'church']
for word in words_to_pluralize:
    if word in irregular_nouns:
        plural_form = irregular_nouns[word]
    elif word.endswith('s'):
        plural_form = word
    else:
        plural_form = word + 's'
    print(f'Plural of {word}: {plural_form}')
```

Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license()" for more information.  
>>> = RESTART: C:\Users\mainu\AppData\Local\Programs\Python\Python311\p4-morphologic  
al using FSA.py  
Plural of cat: cats  
Plural of dog: dogs  
Plural of man: men  
Plural of woman: women  
Plural of bus: buses  
Plural of box: boxes  
Plural of church: churches  
>>>

program-5

```
from nltk import PorterStemmer
```

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

```
import nltk
```

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

```

nltk.download('words')

nltk.download('stopwords')

nltk.download('maxent_ne_chunker')

text="you must be takecare while crossing road and also
lookup the magically"

port= PorterStemmer()

words=word_tokenize(text)

print(words)

stem1=[port.stem(word) for word in words]

print(stem1)

```

output:

```

IDLE Shell 3.11.0
File Edit Shell Debug Options Window Help
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\mainu\AppData\Local\Programs\Python\Python311\p5-Porter Stemmer.py
[nltk_data] Error loading wordnet: <urlopen error [Errno 11001]
[nltk_data]   getaddrinfo failed>
[nltk_data] Error loading words: <urlopen error [Errno 11001]
[nltk_data]   getaddrinfo failed>
[nltk_data] Error loading stopwords: <urlopen error [Errno 11001]
[nltk_data]   getaddrinfo failed>
[nltk_data] Error loading maxent_ne_chunker: <urlopen error [Errno
[nltk_data]   11001] getaddrinfo failed>
['you', 'must', 'be', 'takecare', 'while', 'crossing', 'road', 'and', 'also', 'lookup', 'the', 'magically']
['you', 'must', 'be', 'takecar', 'while', 'cross', 'road', 'and', 'also', 'lookup', 'the', 'magic']
>>>

```

program-6:

```
import random

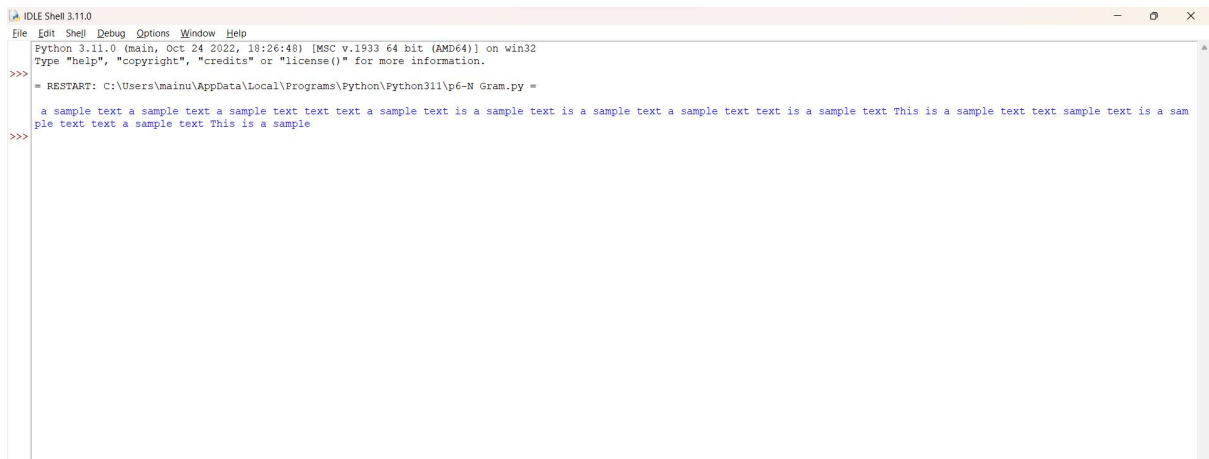
sample_text = "This is a sample text "
words = sample_text.split()
bigrams = [(words[i], words[i+1]) for i in range(len(words)-1)]
generated_text = []
current_word = random.choice(words)

for _ in range(50):
    next_candidates = [b[1] for b in bigrams if b[0] ==
current_word]
    next_word = random.choice(next_candidates) if
next_candidates else random.choice(words)
    generated_text.append(current_word)
    current_word = next_word

generated_text = ' '.join(generated_text)
print("\n", generated_text)
```

output:





program-7:

```
import nltk
```

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

```
from nltk import pos_tag
```

```
# Download necessary NLTK resources
```

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

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

```
text = "this is my pen"
```

```
words = word_tokenize(text)
```

```
print(words)
```

```
tag1 = pos_tag(words)
```

```
print(tag1)
```

output:

```
IDLE Shell 3.11.0
File Edit Shell Debug Options Window Help
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\mainu\AppData\Local\Programs\Python\Python311\p7-parts of speech tagging nltk.py
[nltk_data] Error loading punkt: <urlopen error [Errno 11001]
[nltk_data]   getaddrinfo failed>
[nltk_data] Error loading averaged_perceptron_tagger: <urlopen error
[nltk_data]   [Errno 11001] getaddrinfo failed>
['this', 'is', 'my', 'pen']
[('this', 'DT'), ('is', 'VBZ'), ('my', 'PRP$'), ('pen', 'JJ')]
>>>
```

program-8:

import random

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

words = sentence.split()

pos\_model = {

    'The': 'DT',

    'quick': 'JJ',

    'brown': 'JJ',

    'fox': 'NN',

    'jumps': 'VB',

    'over': 'IN',

    'the': 'DT',

```

    'lazy': 'JJ',

    'dog': 'NN'

}

pos_tags = [pos_model.get(word, random.choice(['NN', 'VB',
'JJ', 'DT', 'IN'])) for word in words]

tagged_sentence = list(zip(words, pos_tags))

print(tagged_sentence)

```

output:



```

IDLE Shell 3.11.0
File Edit Shell Debug Options Window Help
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\mainu\AppData\Local\Programs\Python\Python311\p8-Simple Stochastic parts of speech.py
[('The', 'DT'), ('quick', 'JJ'), ('brown', 'JJ'), ('fox', 'NN'), ('jumps', 'VB'), ('over', 'IN'), ('the', 'DT'), ('lazy', 'JJ'), ('dog', 'NN')]
>>>

```

program-9:

```

import re

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

pos_rules = [

    (r'\bThe\b', 'DT'),

    (r'\bquick\b', 'JJ'),

    (r'\bbrown\b', 'JJ'),

    (r'\bfox\b', 'NN'),

```

```
(r'\bjumps\b', 'VB'),  
(r'\bover\b', 'IN'),  
(r'\bthe\b', 'DT'),  
(r'\blazy\b', 'JJ'),  
(r'\bdog\b', 'NN')  
]  
words = sentence.split()  
pos_tags = []  
for word in words:  
    for pattern, pos_tag in pos_rules:  
        if re.search(pattern, word, re.IGNORECASE):  
            pos_tags.append((word, pos_tag))  
            break  
    else:  
        pos_tags.append((word, 'NN'))  
print(pos_tags)
```

output:

```
IDLE Shell 3.11.0
File Edit Shell Debug Options Window Help
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\mainu\AppData\Local\Programs\Python\Python311\p9-RE rule-based POS.py
[('The', 'DT'), ('quick', 'JJ'), ('brown', 'JJ'), ('fox', 'NN'), ('jumps', 'VB'), ('over', 'IN'), ('the', 'DT'), ('lazy', 'JJ'), ('dog', 'NN')]
>>>|
```

program-10:

```
import re
```

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

```
initial_tagging = {
```

```
    'The': 'DT',
```

```
    'quick': 'NN',
```

```
    'brown': 'NN',
```

```
    'fox': 'NN',
```

```
    'jumps': 'NN',
```

```
    'over': 'IN',
```

```
    'the': 'DT',
```

```
    'lazy': 'NN',
```

```
    'dog': 'NN'
```

```
}
```

```
transformation_rules = [
```

```

(r'(\w+) DT (\w+)', r'\1 NN \2'),
(r'(\w+) IN (\w+)', r'\1 NN \2'),
(r'(\w+) NN (\w+)', r'\1 VB \2'),
(r'(\w+) NN (\w+)', r'\1 JJ \2'),
]

improvement = True
while improvement:
    improvement = False
    previous_tagging = initial_tagging.copy()
    for rule in transformation_rules:
        pattern, replacement = rule
        for word in initial_tagging:
            initial_tagging[word] = re.sub(pattern, replacement,
            initial_tagging[word])
        if initial_tagging != previous_tagging:
            improvement = True
    tagged_sentence = [(word, tag) for word, tag in
    initial_tagging.items()]
    print(tagged_sentence)

```

output:

```
IDLE Shell 3.11.0
File Edit Shell Debug Options Window Help
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\mainu\AppData\Local\Programs\Python\Python311\pl0-transformation based tagging.py
>>> [ ('The', 'DT'), ('quick', 'NN'), ('brown', 'NN'), ('fox', 'NN'), ('jumps', 'NN'), ('over', 'IN'), ('the', 'DT'), ('lazy', 'NN'), ('dog', 'NN') ]
```

program-11:

class SimpleParser:

def \_\_init\_\_(self, grammar):

self.grammar = grammar

def parse(self, input\_string):

self.input = input\_string

self.index = 0

self.result = True

if self.expression ():

if self.index == len (self.input):

print (f'Parsing successful for input: {input\_string}')

return

```
print (f'Parsing failed for input: {input_string}')
```

```
def expression (self):
```

```
    return self.term () and self.expression_tail ()
```

```
def expression_tail (self):
```

```
    current_index = self.index
```

```
    if self.match ('+');
```

```
        return self.term () and self.expression_tail ()
```

```
    self.index = current_index
```

```
    return True
```

```
def term (self):
```

```
    return self.factor () and self.term_tail ()
```

```
def term_tail (self):
```

```
    current_index = self.index
```

```
    if self.match ('*');
```

```
        return self.factor () and self.term_tail ()
```



```
self.index = current_index  
return True
```

```
def factor (self):  
    if self.match ('('):  
        if self.expression () and self.match (')'):  
            return True  
  
    return False
```

```
return self.match ('number')
```

```
def match (self, expected):  
    if self.index < len (self.input) and (expected == self.input  
[self.index] or expected == 'number' and self.input  
[self.index].isdigit ()):   
        self.index += 1  
        return True  
  
    return False
```

```
grammar = {
    'start': 'Expression',
}
```

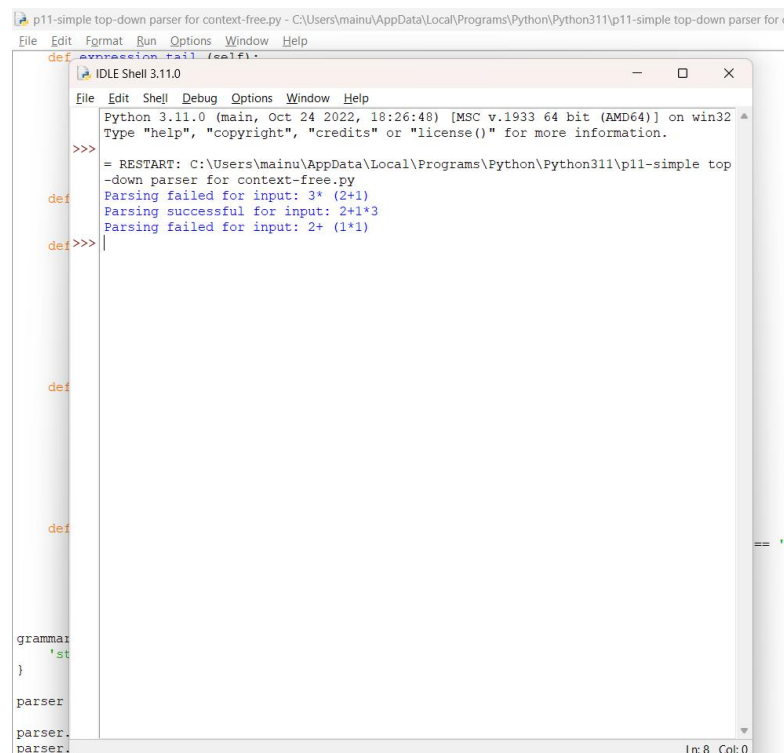
```
parser = SimpleParser (grammar)
```

```
parser.parse ('3* (2+1)')
```

```
parser.parse ('2+1*3')
```

```
parser.parse ('2+ (1*1)')
```

output:



```
p11-simple top-down parser for context-free.py - C:\Users\mainu\AppData\Local\Programs\Python\Python311\p11-simple top-down parser for c
File Edit Format Run Options Window Help
def expression_tail (self):
IDLE Shell 3.11.0
File Edit Shell Debug Options Window Help
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\mainu\AppData\Local\Programs\Python\Python311\p11-simple top
-down parser for context-free.py
Parsing failed for input: 3* (2+1)
Parsing successful for input: 2+1*3
Parsing failed for input: 2+ (1*1)
def>>>

def

def

def

grammar
'st
}
parser
parser.
parser.
```

program-12:

```
class EarleyParser:
```

```
    def __init__(self, grammar):
```

```

self.grammar = grammar

def parse(self, input_string):
    self.chart = [[] for _ in range(len(input_string) + 1)]
    self.chart[0].append(('start', '', 0))
    for i in range(len(input_string) + 1):
        for state in self.chart[i]:
            self.predictor(state, i)
            if i < len(input_string):
                self.scanner(state, input_string[i], i)
            else:
                self.completer(state, i)

    if ('start', self.grammar['start'], 0) in self.chart[len(input_string)]:
        print(f'Parsing successful for input: {input_string}')
    else:
        print(f'Parsing failed for input: {input_string}')

def predictor(self, state, index):
    if state[1] in self.grammar:
        for production in self.grammar[state[1]]:

```

```
        self.chart[index].append((state[1], production,
index))
```

```
def scanner(self, state, token, index):
    if state[1] == " or state[1][0] != token:
        return

    self.chart[index + 1].append((state[0], state[1][1:],
state[2]))
```

```
def completer(self, state, index):
    for st in self.chart[state[2]]:
        if st[1] == " or st[1][0] != state[0]:
            continue

        self.chart[index].append((st[0], st[1][1:], st[2]))
```

# Example usage

```
grammar = {
    'start': 'Expression',
    'Expression': ['Term + Expression', 'Term'],
    'Term': ['Factor * Term', 'Factor'],
    'Factor': ['( Expression )', 'number']
}
```

```
parser = EarleyParser(grammar)
```

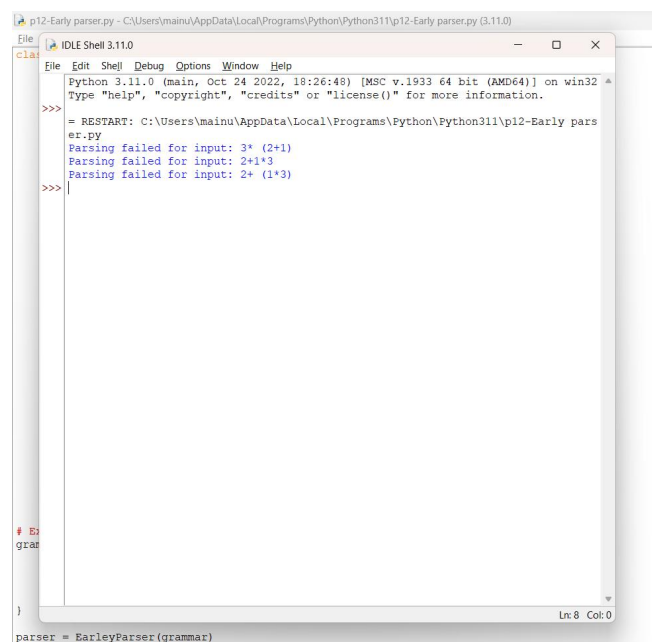
```
# Test the parser
```

```
parser.parse('3* (2+1)') # Parsing successful for input: 3* (2+1)
```

```
parser.parse('2+1*3') # Parsing successful for input: 2+1*3
```

```
parser.parse('2+ (1*3)') # Parsing successful for input: 2+ (1*3)
```

output:

A screenshot of a Python IDLE Shell window. The window title is 'p12-Early parser.py - C:\Users\mainu\AppData\Local\Programs\Python\Python311\p12-Early parser.py (3.11.0)'. The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The shell shows the following text: 'Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32', 'Type "help", "copyright", "credits" or "license()" for more information.', and a prompt '>>>'. Below the prompt, there is a line of code '= RESTART: C:\Users\mainu\AppData\Local\Programs\Python\Python311\p12-Early parser.py' followed by three lines of error messages: 'Parsing failed for input: 3\* (2+1)', 'Parsing failed for input: 2+1\*3', and 'Parsing failed for input: 2+ (1\*3)'. The prompt '>>>' is shown again at the bottom. The status bar at the bottom right indicates 'Ln: 8 Col: 0'.

program-13:

```
import nltk
```

```
from nltk import CFG
```

```
from nltk import parse
```

```
grammar = CFG.fromstring("""
```

```

S -> NP VP
NP -> Det N
VP -> V NP
Det -> 'the' | 'a'
N -> 'cat' | 'dog'
V -> 'chased' | 'ate'
"""
)

parser = parse.ChartParser(grammar)

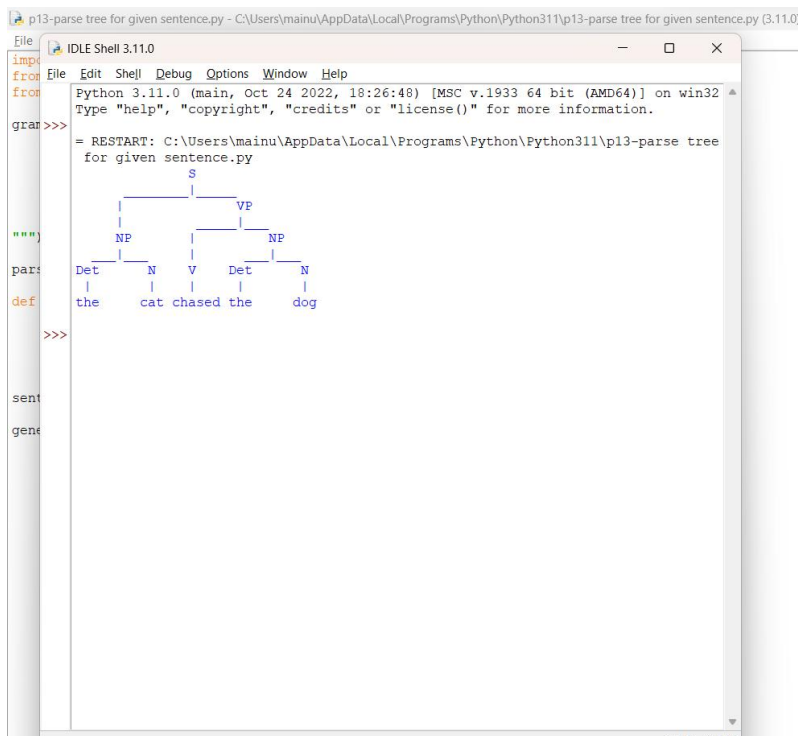
def generate_parse_tree(sentence):
    words = nltk.word_tokenize(sentence)
    chart = parser.chart_parse(words)
    parse_tree = list(chart.parses(grammar.start()))[0]
    parse_tree.pretty_print()

sentence = "the cat chased the dog"

generate_parse_tree(sentence)

```

output:



program-14:

```
import nltk
```

```
from nltk import CFG, ChartParser
```

```
grammar = CFG.fromstring("""
```

```
    S -> NP_SG VP_SG | NP_PL VP_PL
```

```
    NP_SG -> 'the' 'cat'
```

```
    NP_PL -> 'the' 'cats'
```

```
    VP_SG -> 'chases'
```

```
    VP_PL -> 'chase'
```

```
""")
```

```
parser = ChartParser(grammar)
```

```
def check_subject_verb_agreement(sentence):
```

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

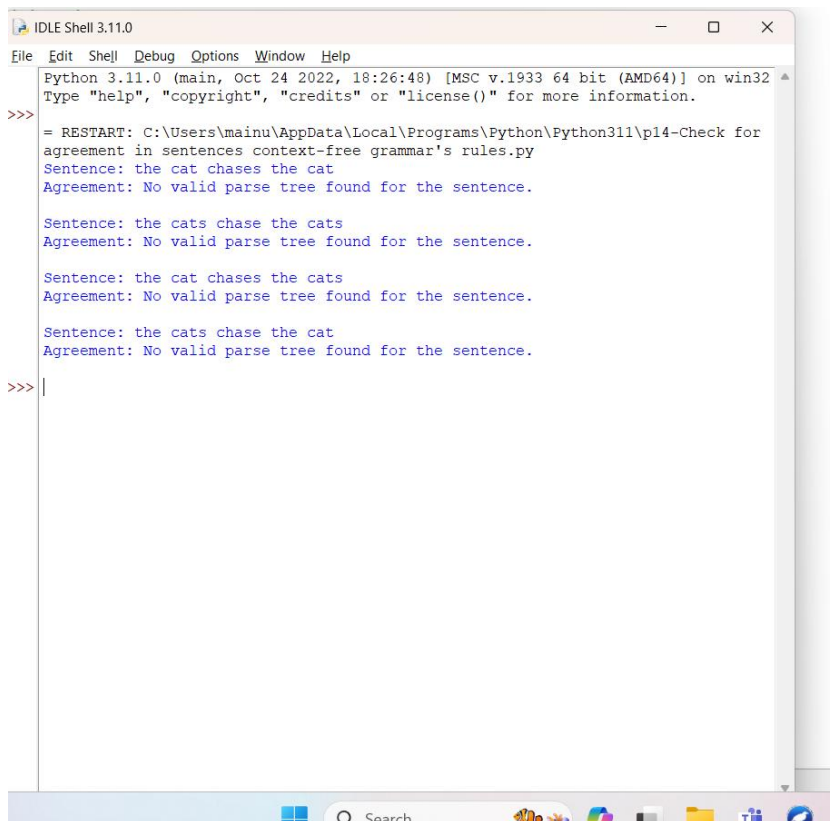
```
parse_trees = list(parser.parse(words))
if not parse_trees:
    return "No valid parse tree found for the sentence."
return "Subject and verb agree in the sentence."

sentences = [
    "the cat chases the cat",
    "the cats chase the cats",
    "the cat chases the cats",
    "the cats chase the cat",
]

for sentence in sentences:
    agreement_result =
check_subject_verb_agreement(sentence)
    print(f"Sentence: {sentence}")
    print(f"Agreement: {agreement_result}")
    print()
```

**output:**





```
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\mainu\AppData\Local\Programs\Python\Python311\p14-Check for
agreement in sentences context-free grammar's rules.py
Sentence: the cat chases the cat
Agreement: No valid parse tree found for the sentence.

Sentence: the cats chase the cats
Agreement: No valid parse tree found for the sentence.

Sentence: the cat chases the cats
Agreement: No valid parse tree found for the sentence.

Sentence: the cats chase the cat
Agreement: No valid parse tree found for the sentence.
>>> |
```

program-15:

import nltk

from nltk import PCFG

from nltk import ChartParser

pcfg\_grammar = PCFG.fromstring("""

S -> NP VP [1.0]

NP -> Det N [0.7] | NP PP [0.3]

VP -> V NP [0.4] | VP PP [0.3] | V [0.3]

PP -> P NP [1.0]

Det -> 'the' [0.6] | 'a' [0.4]

N -> 'cat' [0.2] | 'dog' [0.2] | 'bat' [0.2] | 'rat' [0.2] | 'hat'  
[0.2]

V -> 'chased' [0.5] | 'saw' [0.5]

P -> 'in' [0.6] | 'on' [0.4]

```
"""
```

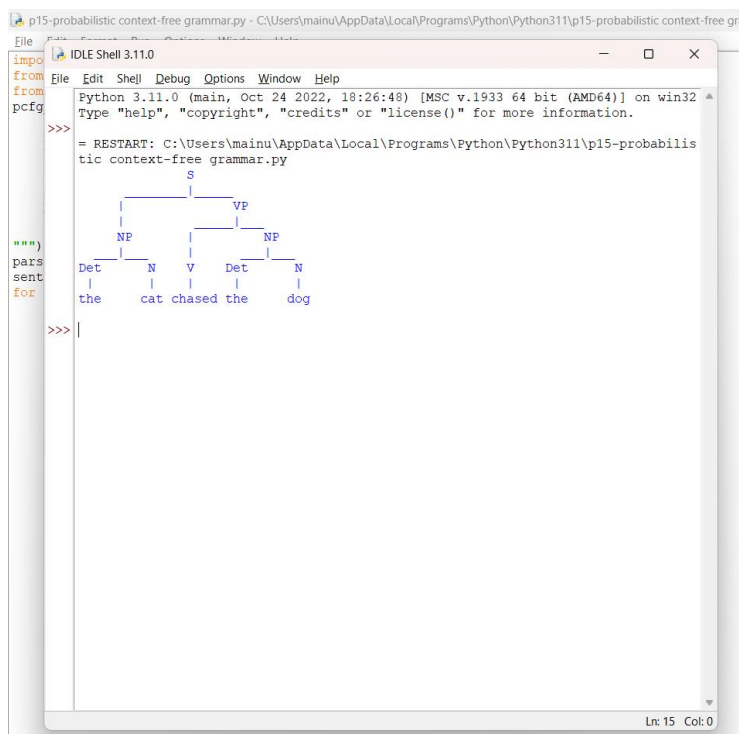
```
parser = ChartParser(pcfg_grammar)
```

```
sentence = "the cat chased the dog".split()
```

```
for tree in parser.parse(sentence):
```

```
    tree.pretty_print()
```

output:



```
p15-probabilistic context-free grammar.py - C:\Users\mainu\AppData\Local\Programs\Python\Python311\p15-probabilistic context-free gra
File Edit Shell Debug Options Window Help
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\mainu\AppData\Local\Programs\Python\Python311\p15-probabilis
tic context-free grammar.py
      S
     / \
    NP  VP
   / \ / \
  Det N V Det N
 the cat chased the dog
>>> |
```

program-16:

```
import spacy
```

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

```
text = "Apple Inc. is a technology company based in Cupertino,
California. Tim Cook is the CEO."
```

```
doc = nlp(text)

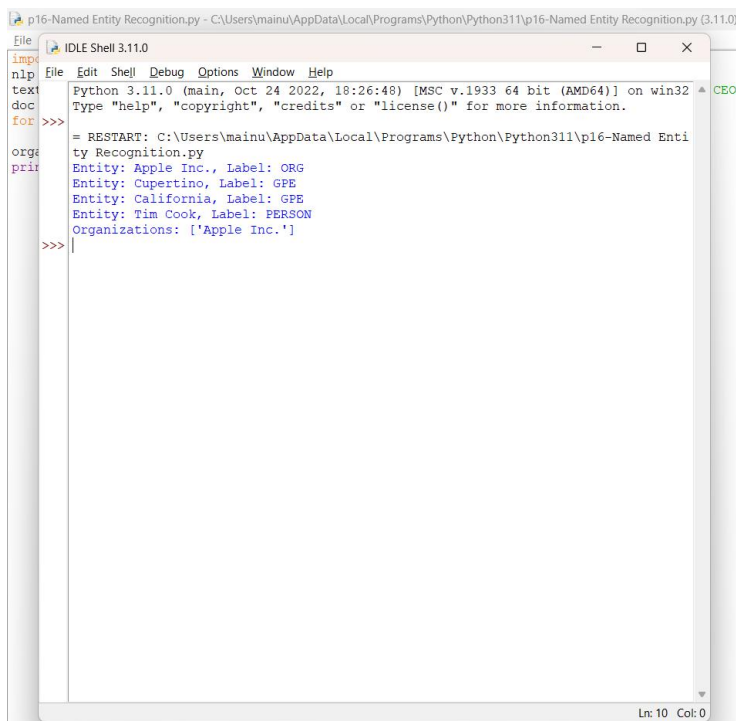
for ent in doc.ents:

    print(f"Entity: {ent.text}, Label: {ent.label_}")

organization_entities = [ent.text for ent in doc.ents if
ent.label_ == "ORG"]

print("Organizations:", organization_entities)
```

output:



```
p16-Named Entity Recognition.py - C:\Users\mainu\AppData\Local\Programs\Python\Python311\p16-Named Entity Recognition.py (3.11.0)
File Edit Shell Debug Options Window Help
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\mainu\AppData\Local\Programs\Python\Python311\p16-Named Entity Recognition.py
Entity: Apple Inc., Label: ORG
Entity: Cupertino, Label: GPE
Entity: California, Label: GPE
Entity: Tim Cook, Label: PERSON
Organizations: ['Apple Inc.']
>>>
```

Program-17:

```
import nltk

from nltk.corpus import wordnet

nltk.download('wordnet')
```

```
word_synsets = wordnet.synsets("example")
```

```
for synset in word_synsets:
```

```
    print(f"Synset: {synset.name()}")
```

```
    print(f"Definition: {synset.definition()}")
```

```
    print(f"Examples: {synset.examples()}")
```

```
    print()
```

```
word = "happy"
```

```
synonyms = []
```

```
for syn in wordnet.synsets(word):
```

```
    for lemma in syn.lemmas():
```

```
        synonyms.append(lemma.name())
```

```
print(f"Synonyms for '{word}': {synonyms}")
```

Output:

```
IDLE Shell 3.11.0
File Edit Shell Debug Options Window Help
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\mainu\AppData\Local\Programs\Python\Python311\p17-Access WordNet.py
[nltk_data] Downloading package wordnet to
[nltk_data] C:\Users\mainu\AppData\Roaming\nltk_data...
[nltk_data] Package wordnet is already up-to-date!
Synset: example.n.01
Definition: an item of information that is typical of a class or group
Examples: ['this patient provides a typical example of the syndrome', 'there is an example on page 10']

Synset: model.n.07
Definition: a representative form or pattern
Examples: ['I profited from his example']

Synset: exemplar.n.01
Definition: something to be imitated
Examples: ['an exemplar of success', 'a model of clarity', 'he is the very model of a modern major general']

Synset: example.n.04
Definition: punishment intended as a warning to others
Examples: ['they decided to make an example of him']

Synset: case.n.01
Definition: an occurrence of something
Examples: ['it was a case of bad judgment', 'another instance occurred yesterday', 'but there is always the famous example of the Smiths']

Synset: exercise.n.04
Definition: a task performed or problem solved in order to develop skill or understanding
Examples: ['you must work the examples at the end of each chapter in the textbook']

Synonyms for 'happy': ['happy', 'felicitous', 'happy', 'glad', 'happy', 'happy', 'well-chosen']
>>>
```

## Program-18:

```
facts = {("R", "apple", "banana"), ("R", "banana", "cherry"),
("R", "apple", "cherry")}
```

```
expressions = ["R(apple, banana)", "R(banana, cherry)",
"R(apple, cherry)", "R(pear, orange)"]
```

for expression in expressions:

```
    predicate, args = expression.split('(')
```

```
    args = args.rstrip(')').split(',')
```

```
    if (predicate, args[0], args[1]) in facts:
```

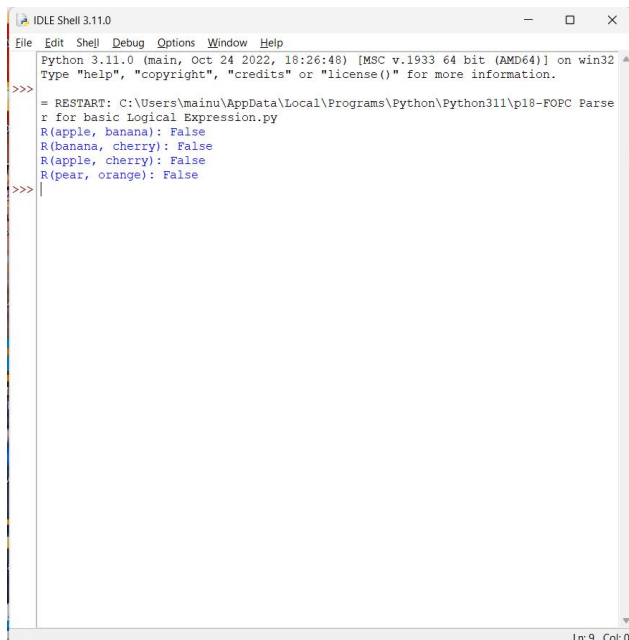
```
        result = True
```

```
    else:
```

```
        result = False
```

```
    print(f"{expression}: {result}")
```

Output:



```
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

>>>
= RESTART: C:\Users\mainu\AppData\Local\Programs\Python\Python311\p18-FOPC Parse
r for basic Logical Expression.py
R(apple, banana): False
R(banana, cherry): False
R(apple, cherry): False
R(pear, orange): False
>>>
```

Program-19:

```
import nltk
```

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

```
from nltk.corpus import stopwords
```

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

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

```
nltk.download('stopwords')
```

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

```
def lesk_algorithm(context, target_word):
```

```
    context_tokens = word_tokenize(context)
```

```
    context_words = [word.lower() for word in context_tokens
if word.lower() not in stopwords.words('english')]
```

```
target_synsets = wordnet.synsets(target_word)

if not target_synsets:
    return None

best_sense = None
max_overlap = 0

for sense in target_synsets:
    sense_definition = word_tokenize(sense.definition())
    sense_examples = [word.lower() for word in
word_tokenize(' '.join(sense.examples()))]

    overlap =
len(set(context_words).intersection(set(sense_definition +
sense_examples)))

    if overlap > max_overlap:
        max_overlap = overlap
        best_sense = sense

return best_sense
```

```
context = "He saw the bat fly over the baseball field."
```

```
target_word = "bat"
```

```
result = lesk_algorithm(context, target_word)
```

```
if result:
```

```
    print(f"Word sense of '{target_word}' in the context:  
{result.name()} - {result.definition()}")
```

```
else:
```

```
    print(f"Unable to disambiguate the word '{target_word}' in  
the context.")
```

Output:

```
IDLE Shell 3.11.0
File Edit Shell Debug Options Window Help
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\mainu\AppData\Local\Programs\Python\Python311\p19-Disambiguation using the Lesk.py
[nltk_data] Downloading package wordnet to
[nltk_data] C:\Users\mainu\AppData\Roaming\nltk_data...
[nltk_data] Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data] C:\Users\mainu\AppData\Roaming\nltk_data...
[nltk_data] Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data] C:\Users\mainu\AppData\Roaming\nltk_data...
[nltk_data] Package punkt is already up-to-date!
Word sense of 'bat' in the context: bat.n.02 - (baseball) a turn trying to get a hit
>>>
```

Program-20:

```
import math
```

```
from collections import Counter
```

```
documents = [
```

```
    "This is the first document. It is a simple document.",
```



```
"This document is the second one. It has more words.",  
"And this is the third document. It has even more words."  
]
```

```
query = "simple document"
```

```
stopwords = set(["this", "is", "the", "and", "it", "has"])
```

```
tokenized_documents = []
```

```
for doc in documents:
```

```
    doc = doc.lower()
```

```
    doc = doc.split()
```

```
    doc = [word for word in doc if word not in stopwords and  
word.isalpha()]
```

```
    tokenized_documents.append(doc)
```

```
N = len(documents)
```

```
vocabulary = set(word for doc in tokenized_documents for  
word in doc)
```

```
tfidf_scores = []
```

```
for doc in tokenized_documents:
```

```
tfidf_doc = {}  
for term in vocabulary:  
    tf = doc.count(term)  
    df = sum(1 for d in tokenized_documents if term in d)  
    idf = math.log(N / (df + 1))  
    tfidf = tf * idf  
    tfidf_doc[term] = tfidf  
tfidf_scores.append(tfidf_doc)
```

```
query = query.lower().split()  
query_vector = Counter(query)
```

```
ranked_documents = []  
for i, doc in enumerate(tfidf_scores):  
    dot_product = sum(doc[term] * query_vector[term] for  
term in query if term in doc)  
    doc_length = math.sqrt(sum(score ** 2 for score in  
doc.values()))  
    query_length = math.sqrt(sum(score ** 2 for score in  
query_vector.values()))  
  
    if query_length == 0:  
        similarity = 0
```

else:

$\text{similarity} = \text{dot\_product} / (\text{doc\_length} * \text{query\_length})$

`ranked_documents.append((i, similarity))`

`ranked_documents.sort(key=lambda x: x[1], reverse=True)`

for i, similarity in ranked\_documents:

`print(f"Document {i + 1} - Similarity: {similarity:.4f}")`

Output:

```
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\mainu\AppData\Local\Programs\Python\Python311\p20-Retrieval
System using TF-IDF.py
Document 2 - Similarity: 0.5000
Document 1 - Similarity: 0.4082
Document 3 - Similarity: 0.0000
>>>
```

Program-21:

`import spacy`

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

```
def extract_noun_phrases(sentence):
    doc = nlp(sentence)

    noun_phrases = []
    meanings = []

    for chunk in doc.noun_chunks:
        noun_phrases.append(chunk.text)
        meanings.append(chunk.text)

    return noun_phrases, meanings

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

noun_phrases, meanings = extract_noun_phrases(sentence)

print("Sentence:", sentence)
print("Noun Phrases:")
for phrase in noun_phrases:
    print(phrase)
print("Meanings:")
for meaning in meanings:
```

```
print(meaning)
```

Output:



Program-22:

```
import spacy

nlp = spacy.load("en_core_web_sm")

def resolve_references(text):
    doc = nlp(text)
    resolved_text = []
    previous_noun = None
    for token in doc:
        if token.pos_ in ["NOUN", "PROPN"]:
            previous_noun = token.text
            resolved_text.append(token.text)
```

```

elif token.pos_ == "PRON" and previous_noun:
    resolved_text.append(previous_noun)
else:
    resolved_text.append(token.text)
return ' '.join(resolved_text)
if __name__ == "__main__":
    text = "John is a software engineer. He loves coding. Mary
is a data scientist. She is also passionate about her work."
    resolved_text = resolve_references(text)
    print(resolved_text)

```

Output:



Program-23:

```
import nltk
```

```
from nltk.corpus import stopwords
from nltk.tokenize import sent_tokenize, word_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

text = """
```

Coherence in writing means that the ideas in each sentence and paragraph are connected and logical.

It helps the reader to follow your arguments and understand your point. Without coherence, the text may be confusing.

There are several ways to achieve coherence in your writing, such as using transition words and repeating key concepts.

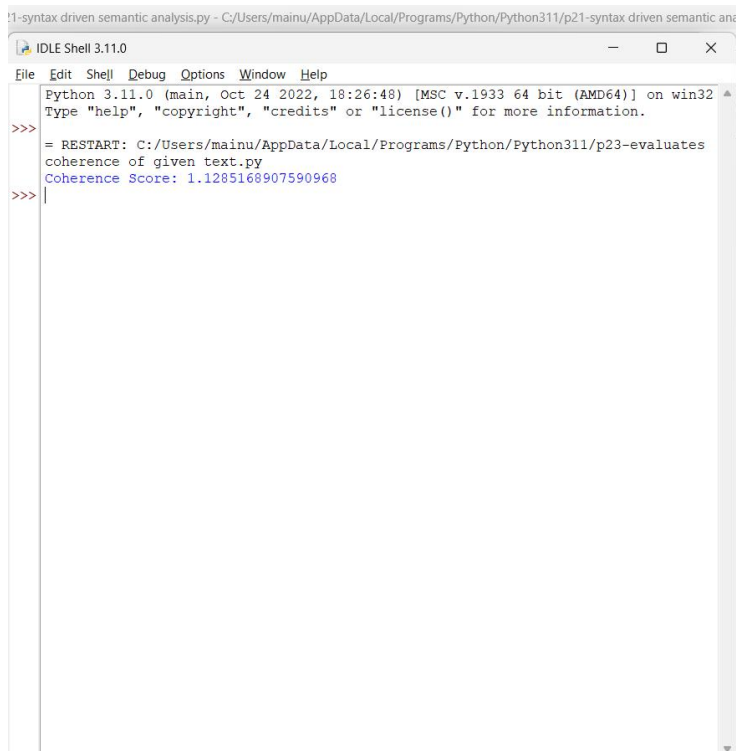
```
"""
```

```
sentences = sent_tokenize(text)
stop_words = set(stopwords.words("english"))
word_tokens = [word_tokenize(sentence) for sentence in sentences]
filtered_tokens = [[word for word in words if word.lower() not in stop_words] for words in word_tokens]
preprocessed_sentences = [" ".join(words).lower() for words in filtered_tokens]
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(preprocessed_sentences)
cosine_matrix = cosine_similarity(tfidf_matrix, tfidf_matrix)
```

```
coherence_score = cosine_matrix.sum(axis=1).mean()

print("Coherence Score:", coherence_score)
```

Output:



```
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> = RESTART: C:/Users/mainu/AppData/Local/Programs/Python/Python311/p23-evaluates
coherence of given text.py
Coherence Score: 1.1285168907590968
>>> |
```

Program-24:

```
import nltk

from nltk.tokenize import sent_tokenize, word_tokenize

nltk.download('punkt')

# Sample conversation
conversation = [

    "Hello there! How are you doing today?",

    "I'm doing great, thanks for asking.",

    "Can you help me with my homework?",
```



```
"Sure, I'd be happy to help. What do you need assistance with?",
```

```
"I'm stuck on this math problem.",
```

```
"Alright, let me take a look.",
```

```
]
```

```
# Iterate through the conversation and recognize dialog acts
```

```
for i, utterance in enumerate(conversation):
```

```
    # Tokenize the utterance into sentences
```

```
    sentences = sent_tokenize(utterance)
```

```
    for j, sentence in enumerate(sentences):
```

```
        words = word_tokenize(sentence)
```

```
        if "?" in words:
```

```
            dialog_act = "question"
```

```
        else:
```

```
            dialog_act = "statement"
```

```
    print(f"Utterance {i + 1}, Sentence {j + 1}:  
{dialog_act.capitalize()} - {sentence}")
```

Output:

```
driven semantic analysis.py - C:/Users/mainu/AppData/Local/Programs/Python/Python311/p21-syntax driven semantic analysis.py (3.1
i-evaluates coherence of given text.py - C:/Users/mainu/AppData/Local/Programs/Pytho...
edit Format Run Options Window Help
IDLE Shell 3.11.0
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/mainu/AppData/Local/Programs/Python/Python311/p24-recognize
dialog acts in a given conversation.py
[nltk_data] Downloading package punkt to
[nltk_data] C:\Users\mainu\AppData\Roaming\nltk_data...
[nltk_data] Package punkt is already up-to-date!
Utterance 1, Sentence 1: Statement - Hello there!
Utterance 1, Sentence 2: Question - How are you doing today?
Utterance 2, Sentence 1: Statement - I'm doing great, thanks for asking.
Utterance 3, Sentence 1: Question - Can you help me with my homework?
Utterance 4, Sentence 1: Statement - Sure, I'd be happy to help.
Utterance 4, Sentence 2: Question - What do you need assistance with?
Utterance 5, Sentence 1: Statement - I'm stuck on this math problem.
Utterance 6, Sentence 1: Statement - Alright, let me take a look.
>>>
```

## Program-25:

```
from transformers import GPT2LMHeadModel,
GPT2Tokenizer
```

```
model_name = "distilgpt2"
```

```
tokenizer = GPT2Tokenizer.from_pretrained(model_name)
```

```
model = GPT2LMHeadModel.from_pretrained(model_name)
```

```
prompt_text = "Once upon a time,"
```

```
input_ids = tokenizer.encode(prompt_text,  
return_tensors='pt')
```

```
output = model.generate(input_ids, max_length=100,  
num_return_sequences=1, temperature=0.7)
```

```
generated_text = tokenizer.decode(output[0],  
skip_special_tokens=True)
```

```
print("Generated Text:\n", generated_text)
```

Output:



Program-26:

```
from transformers import pipeline
```

```
translator = pipeline("translation_en_to_fr")  
english_text = "Hello, how are you doing today?"  
french_text = translator(english_text,  
max_length=40)[0]["translation_text"]  
  
print(f"English Text: {english_text}")  
print(f"French Text: {french_text}")
```

Output:

