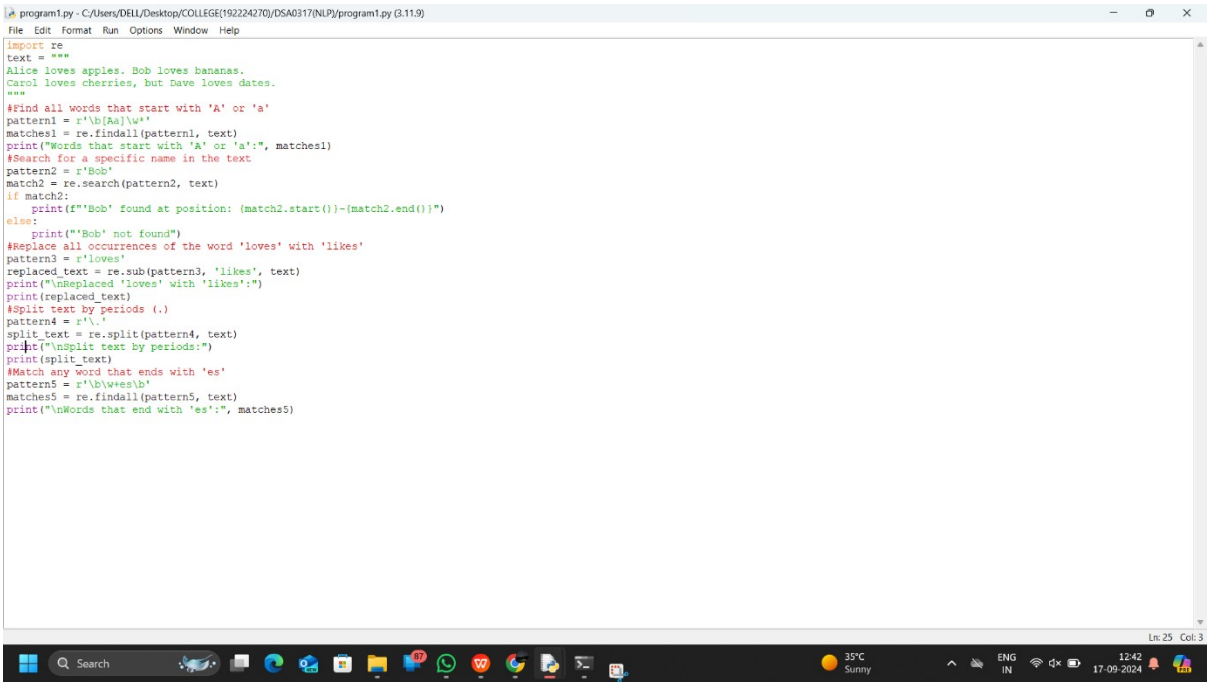
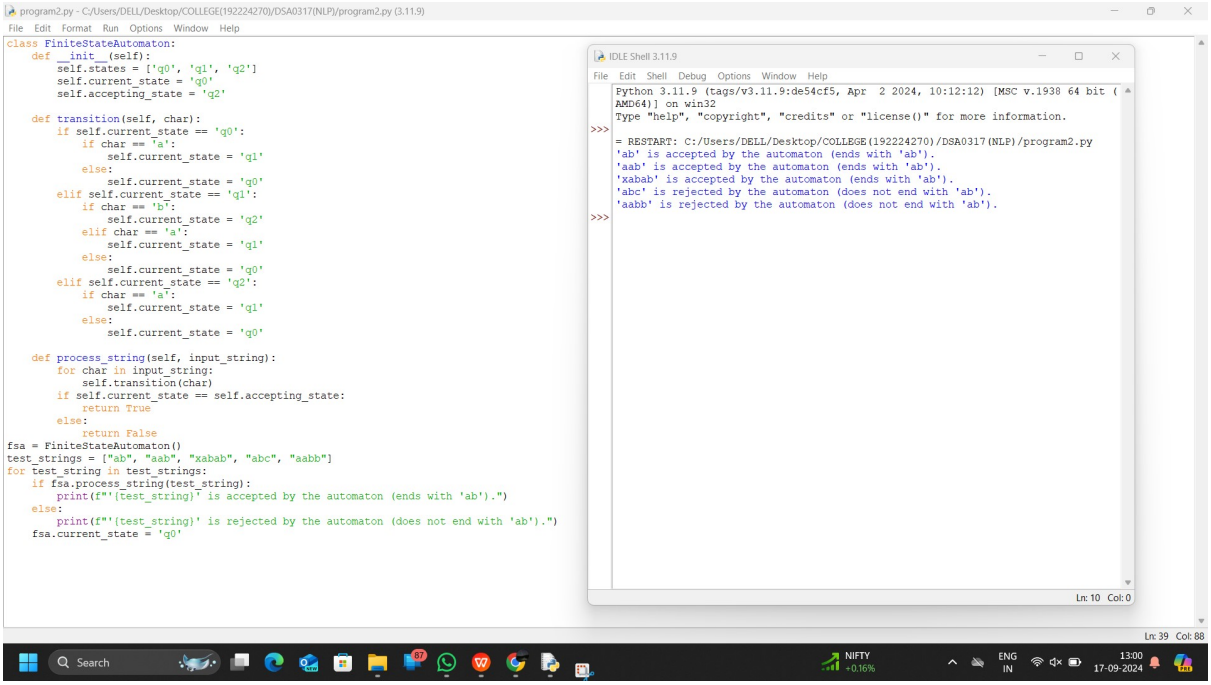
AARON PULIKKOTTIL(192224270)

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



```
import re
text = """
Alice loves apples. Bob loves bananas.
Carol loves cherries, but Dave loves dates.
"""
#Find all words that start with 'A' or 'a'
pattern1 = r'\b[Aa]\w*'
matches1 = re.findall(pattern1, text)
print("Words that start with 'A' or 'a':", matches1)
#Search for a specific name in the text
pattern2 = r'Bob'
match2 = re.search(pattern2, text)
if match2:
    print(f"'Bob' found at position: {match2.start()}-{match2.end()}")
else:
    print("'Bob' not found")
#Replace all occurrences of the word 'loves' with 'likes'
pattern3 = r'loves'
replaced_text = re.sub(pattern3, 'likes', text)
print("\nReplaced 'loves' with 'likes':")
print(replaced_text)
#Split text by periods (.)
pattern4 = r'\.'
split_text = re.split(pattern4, text)
print("\nSplit text by periods:")
print(split_text)
#Match any word that ends with 'es'
pattern5 = r'\b\w+es\b'
matches5 = re.findall(pattern5, text)
print("\nWords that end with 'es':", matches5)
```

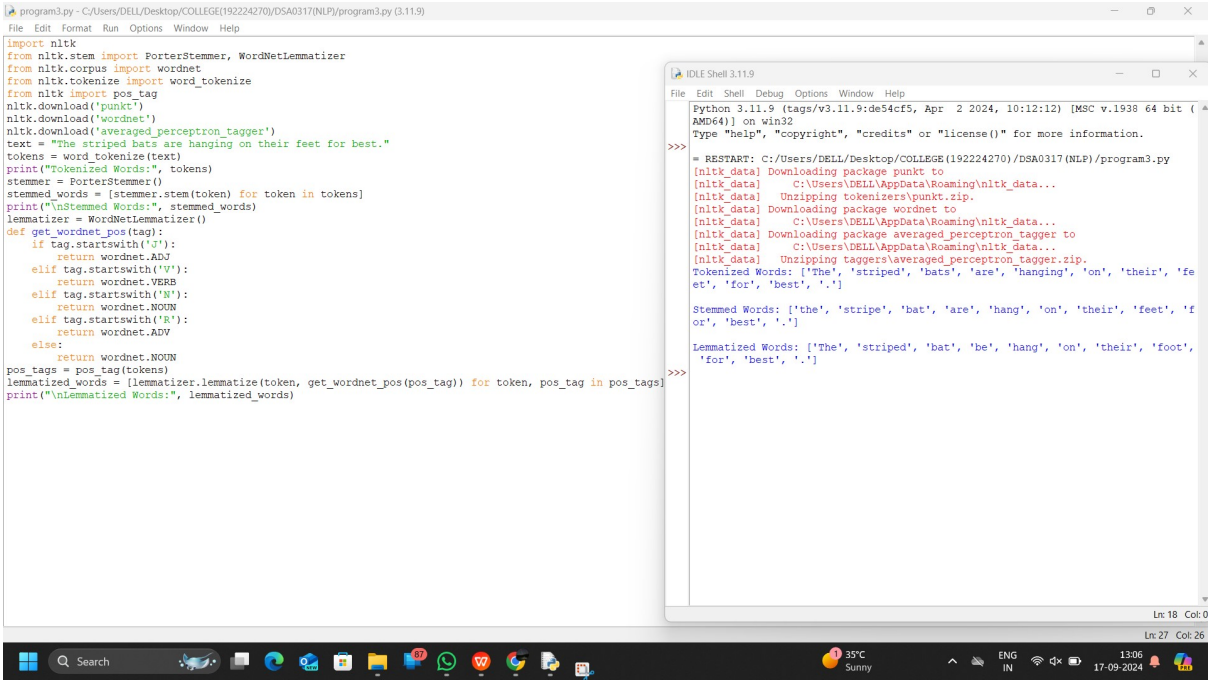
2) Implement a basic finite state automaton that recognizes a specific language or pattern. In this example, we'll create a simple automaton to match strings ending with 'ab' using python.



```
class FiniteStateAutomaton:
    def __init__(self):
        self.states = ['q0', 'q1', 'q2']
        self.current_state = 'q0'
        self.accepting_state = 'q2'

    def transition(self, char):
        if self.current_state == 'q0':
            if char == 'a':
                self.current_state = 'q1'
            else:
                self.current_state = 'q0'
        elif self.current_state == 'q1':
            if char == 'b':
                self.current_state = 'q2'
            elif char == 'a':
                self.current_state = 'q1'
            else:
                self.current_state = 'q0'
        elif self.current_state == 'q2':
            if char == 'a':
                self.current_state = 'q1'
            else:
                self.current_state = 'q0'

    def process_string(self, input_string):
        for char in input_string:
            self.transition(char)
        if self.current_state == self.accepting_state:
            return True
        else:
            return False
fsa = FiniteStateAutomaton()
test_strings = ["ab", "aab", "xabab", "abc", "aabb"]
for test_string in test_strings:
    if fsa.process_string(test_string):
        print(f"'{test_string}' is accepted by the automaton (ends with 'ab').")
    else:
        print(f"'{test_string}' is rejected by the automaton (does not end with 'ab').")
    fsa.current_state = 'q0'
```

```
Python 3.11.9 (tags/v3.11.9:de54cf5, Apr  2 2024, 10:12:12) [MSC v.1938 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/DELL/Desktop/COLLEGE(192224270)/DSA0317(NLP)/program2.py
'ab' is accepted by the automaton (ends with 'ab').
'aab' is accepted by the automaton (ends with 'ab').
'xabab' is accepted by the automaton (ends with 'ab').
'abc' is rejected by the automaton (does not end with 'ab').
'aabb' is rejected by the automaton (does not end with 'ab').
>>>
```

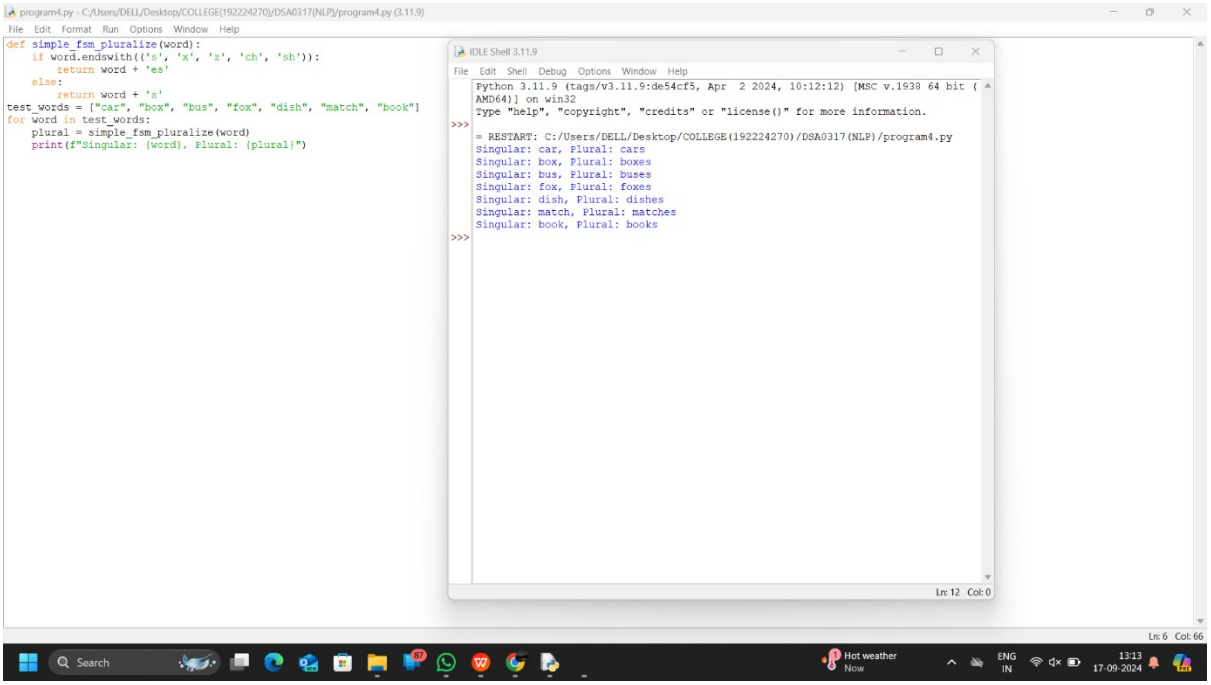
3) Write program demonstrates how to perform morphological analysis using the NLTK library in Python.



```
import nltk
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk.corpus import wordnet
from nltk.tokenize import word_tokenize
from nltk import pos_tag
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
text = "The striped bats are hanging on their feet for best."
tokens = word_tokenize(text)
print("Tokenized Words:", tokens)
stemmer = PorterStemmer()
stemmed_words = [stemmer.stem(token) for token in tokens]
print("\nStemmed Words:", stemmed_words)
lemmatizer = WordNetLemmatizer()
def get_wordnet_pos(tag):
    if tag.startswith('J'):
        return wordnet.ADJ
    elif tag.startswith('V'):
        return wordnet.VERB
    elif tag.startswith('N'):
        return wordnet.NOUN
    elif tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN
pos_tags = pos_tag(tokens)
lemmatized_words = [lemmatizer.lemmatize(token, get_wordnet_pos(pos_tag)) for token, pos_tag in pos_tags]
print("\nLemmatized Words:", lemmatized_words)
```

```
Python 3.11.9 (tags/v3.11.9:de54cf5, Apr  2 2024, 10:12:12) [MSC v.1938 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/DELL/Desktop/COLLEGE(192224270)/DSA0317(NLP)/program3.py
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\DELL\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping tokenizers\punkt.zip.
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\DELL\AppData\Roaming\nltk_data...
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\DELL\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping taggers\averaged_perceptron_tagger.zip.
Tokenized Words: ['The', 'striped', 'bats', 'are', 'hanging', 'on', 'their', 'fe
et', 'for', 'best', '.']

Stemmed Words: ['the', 'stripe', 'bat', 'are', 'hang', 'on', 'their', 'feet', 'f
or', 'best', '.']

Lemmatized Words: ['The', 'striped', 'bat', 'be', 'hang', 'on', 'their', 'foot',
 'for', 'best', '.']
>>>
```

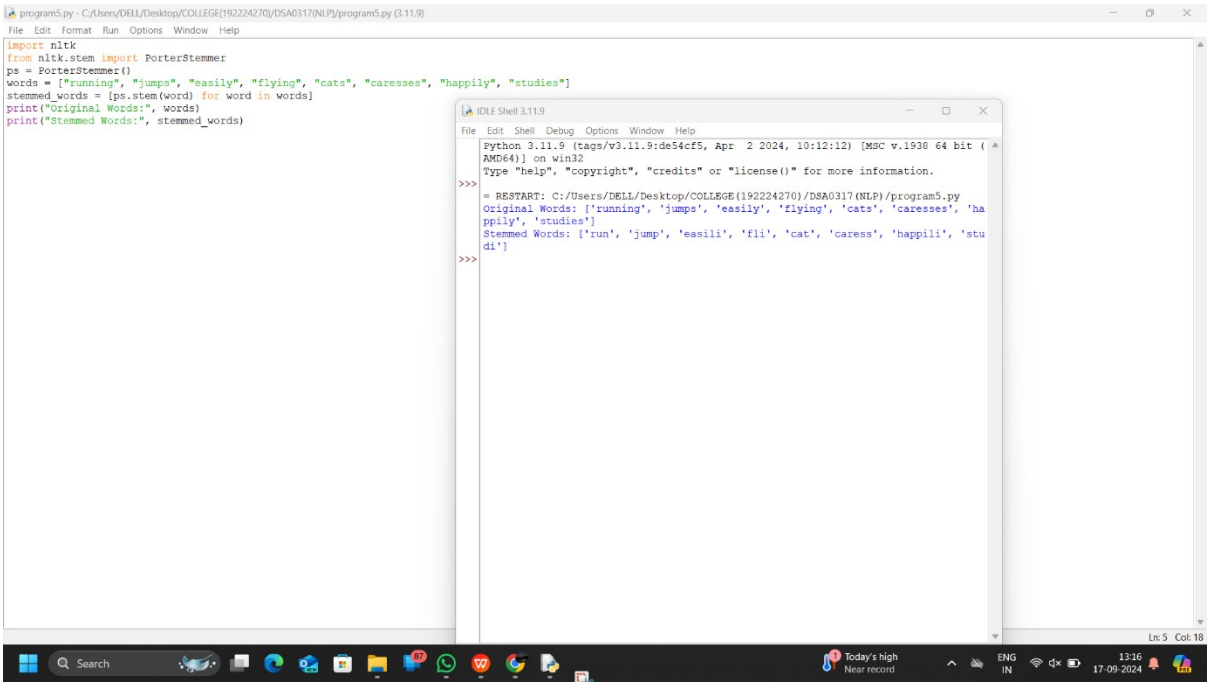
4) Implement a finite-state machine for morphological parsing. In this example, we'll create a simple machine to generate plural forms of English nouns using python.



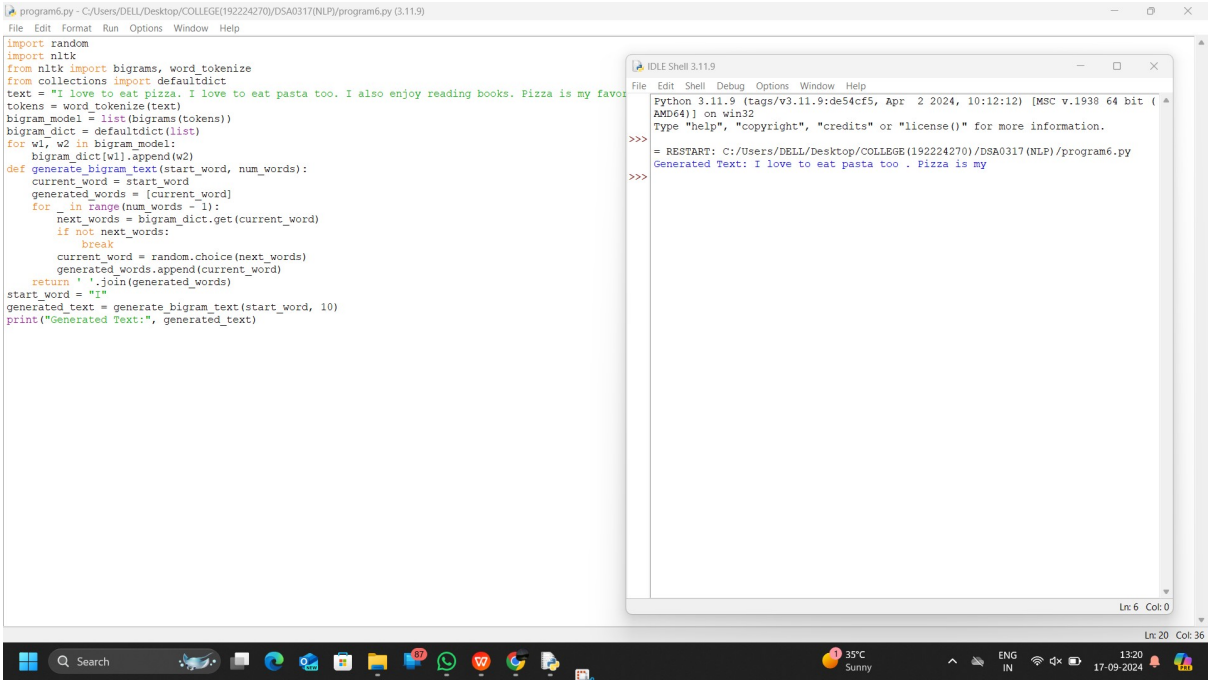
```
def simple_fsm_pluralize(word):
    if word.endswith(('s', 'x', 'z', 'ch', 'sh')):
        return word + 'es'
    else:
        return word + 's'
test_words = ["car", "box", "bus", "fox", "dish", "match", "book"]
for word in test_words:
    plural = simple_fsm_pluralize(word)
    print(f"Singular: {word}, Plural: {plural}")
```

```
Python 3.11.9 (tags/v3.11.9:de54cf5, Apr  2 2024, 10:12:12) [MSC v.1938 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/DELL/Desktop/COLLEGE(192224270)/DSA0317(NLP)/program4.py
Singular: car, Plural: cars
Singular: box, Plural: boxes
Singular: bus, Plural: buses
Singular: fox, Plural: foxes
Singular: dish, Plural: dishes
Singular: match, Plural: matches
Singular: book, Plural: books
>>>
```

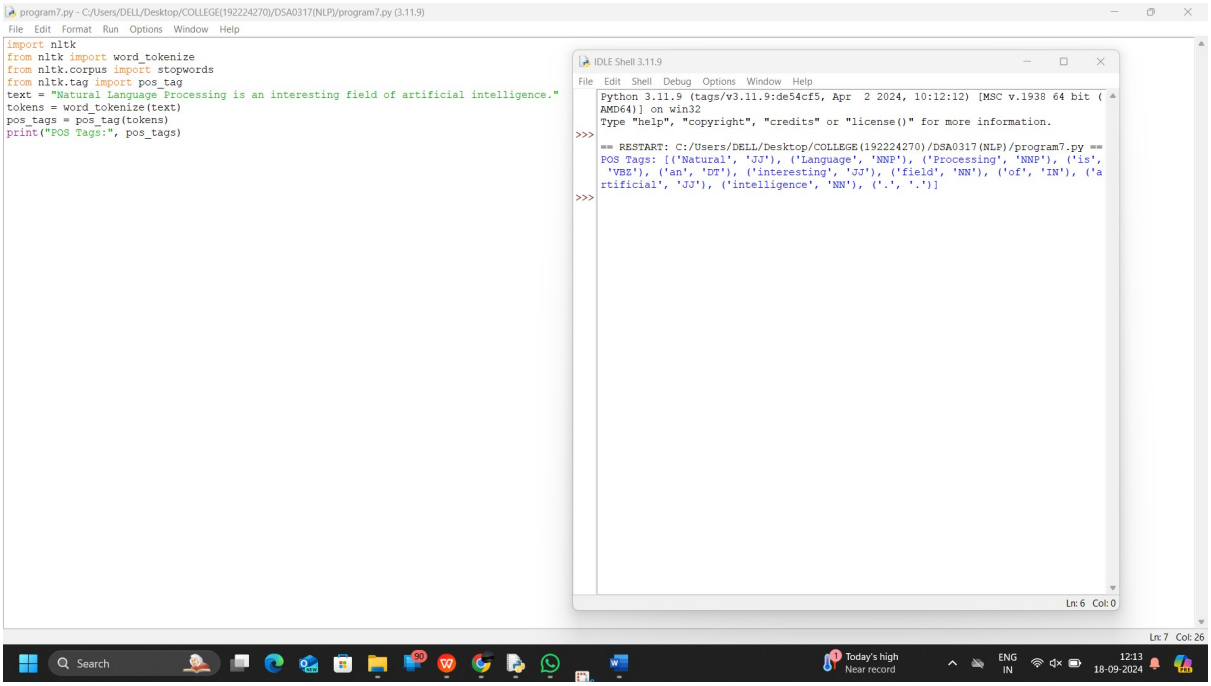
5) Use the Porter Stemmer algorithm to perform word stemming on a list of words using python libraries.



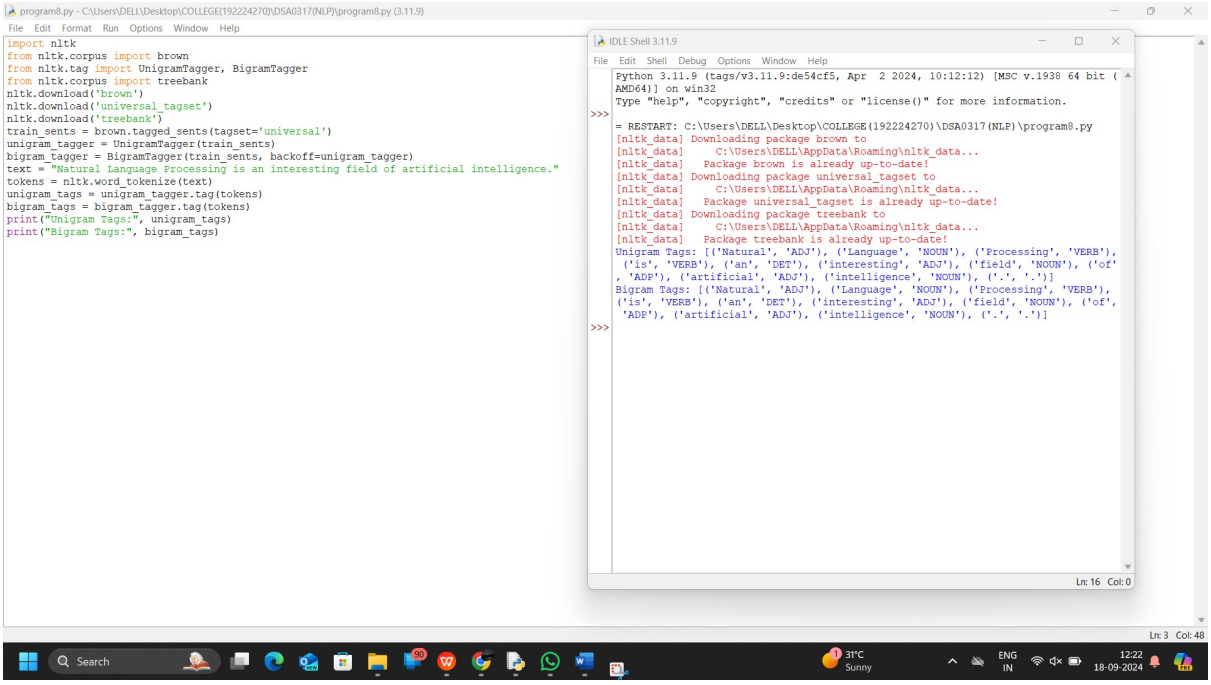
6) Implement a basic N-gram model for text generation. For example, generate text using a bigram model using python.



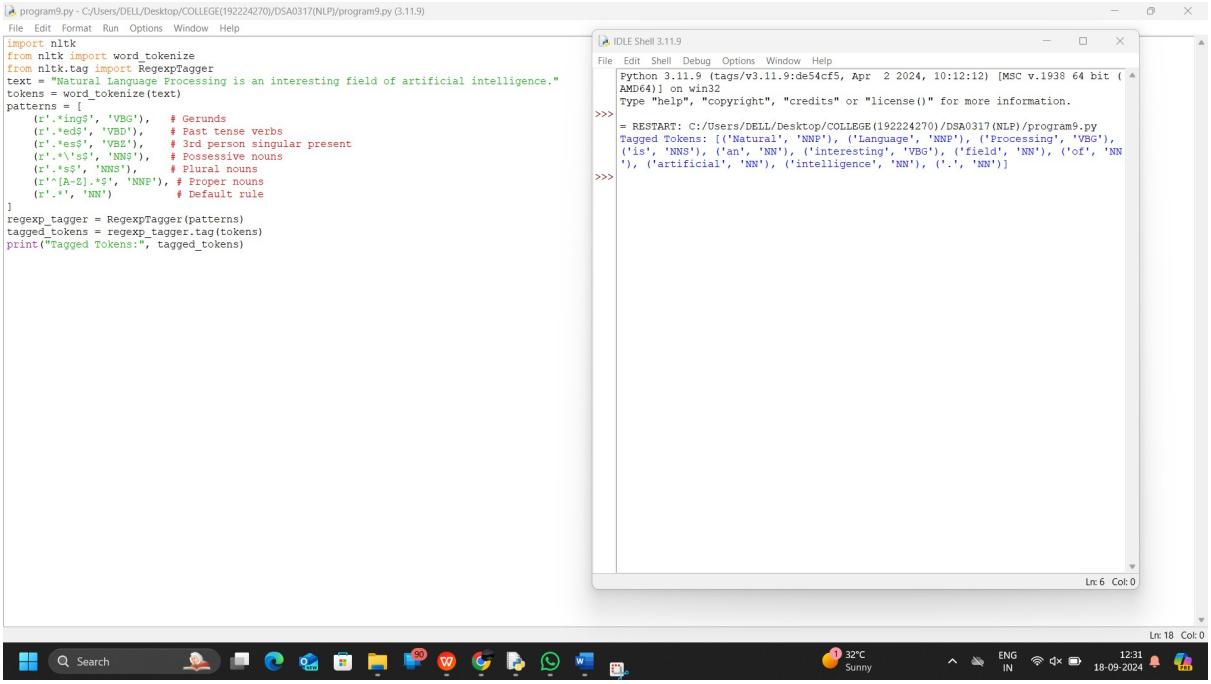
7) Write program using the NLTK library to perform part-of-speech tagging on a text.



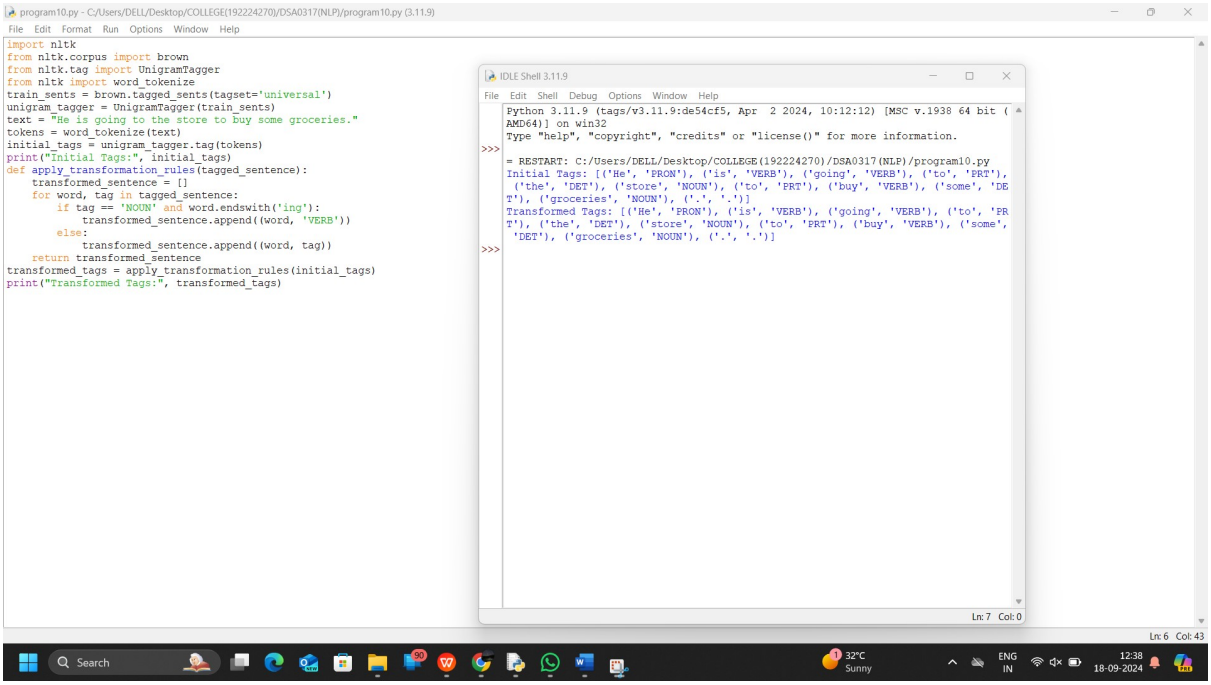
8) Implement a simple stochastic part-of-speech tagging algorithm using a basic probabilistic model to assign POS tags using python.



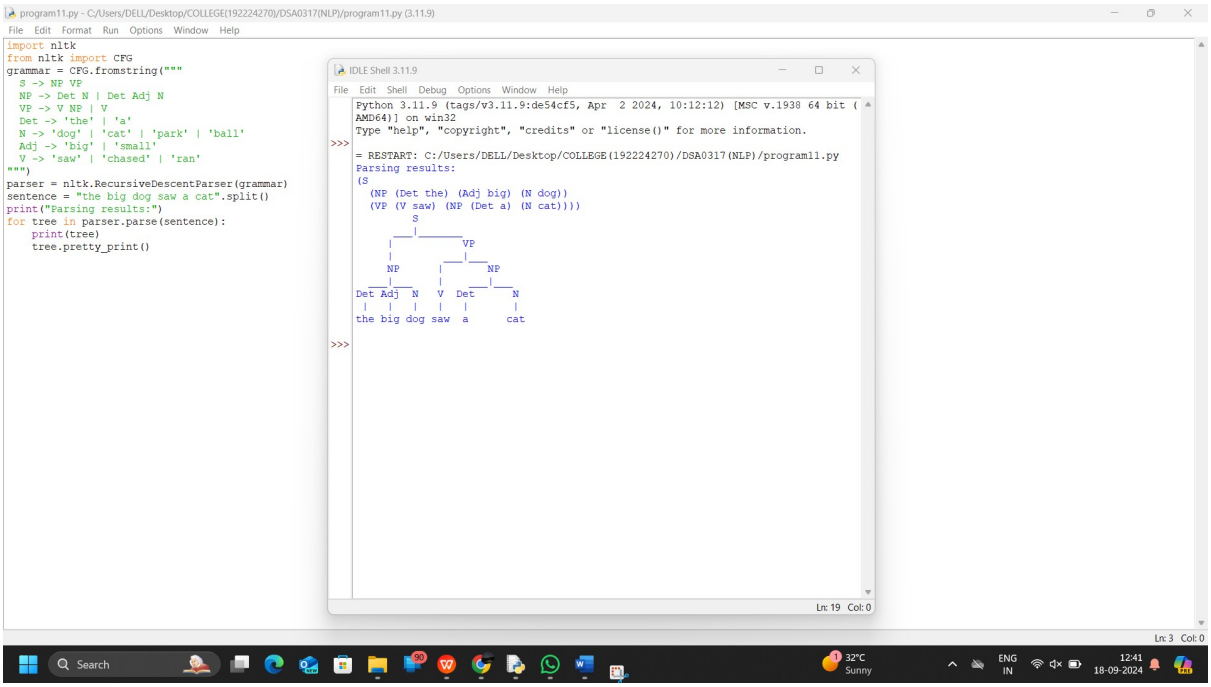
9) Implement a rule-based part-of-speech tagging system using regular expressions using python.



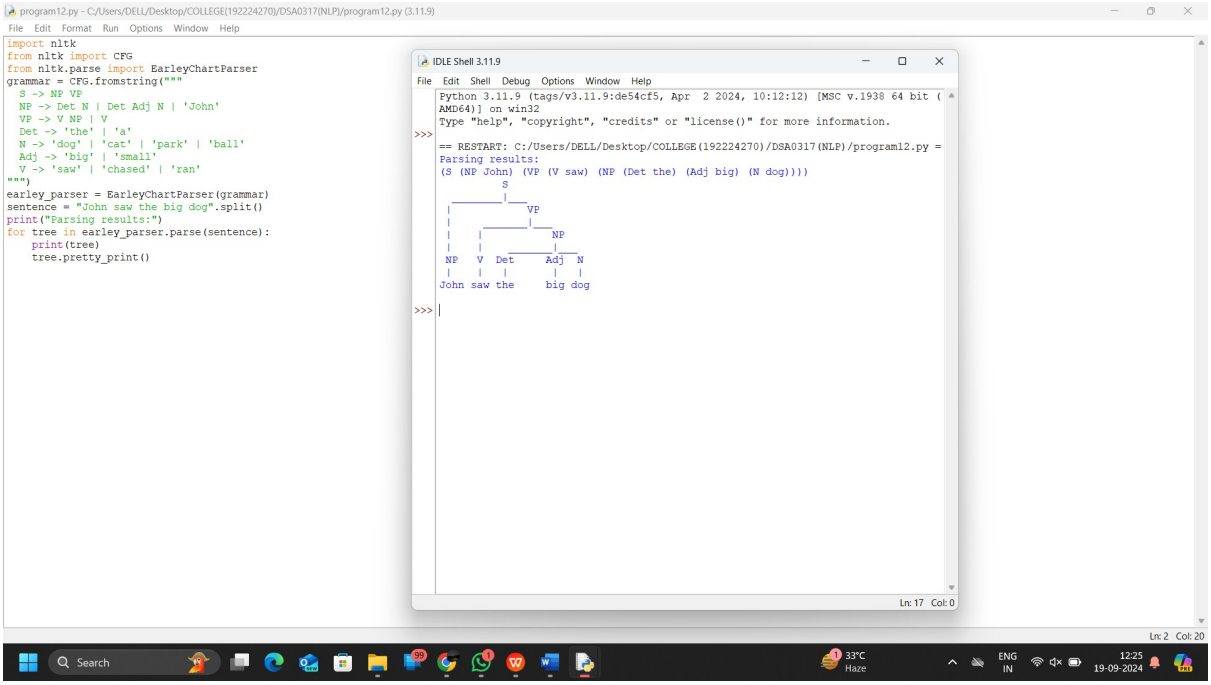
10) Implement transformation-based tagging using a set of transformation rules, apply a simple rule to tag words using python.



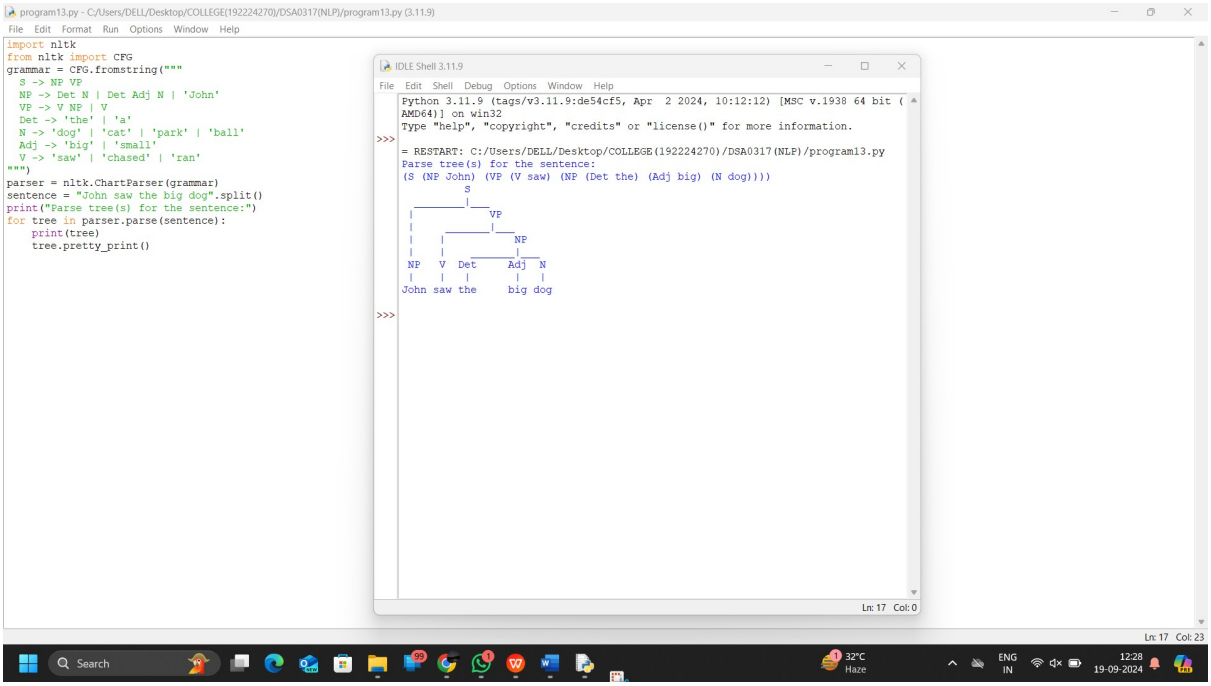
11) Implement a simple top-down parser for context-free grammars using python.



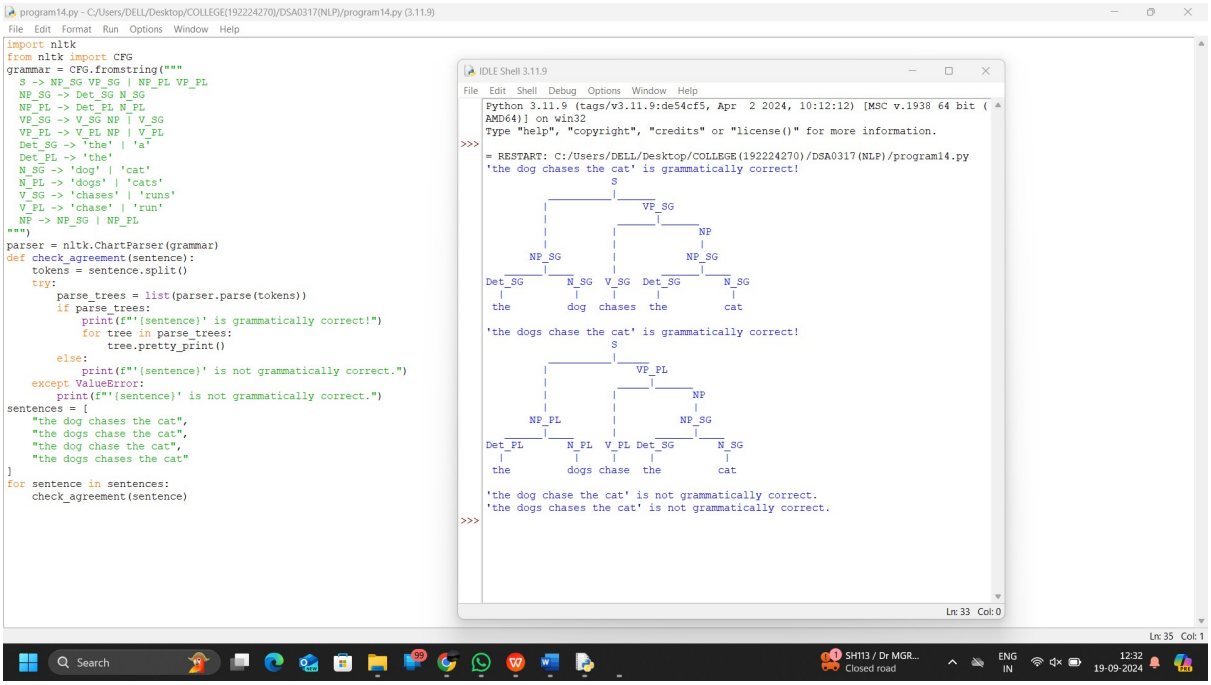
12) Implement an Earley parser for context-free grammars using a simple python program.



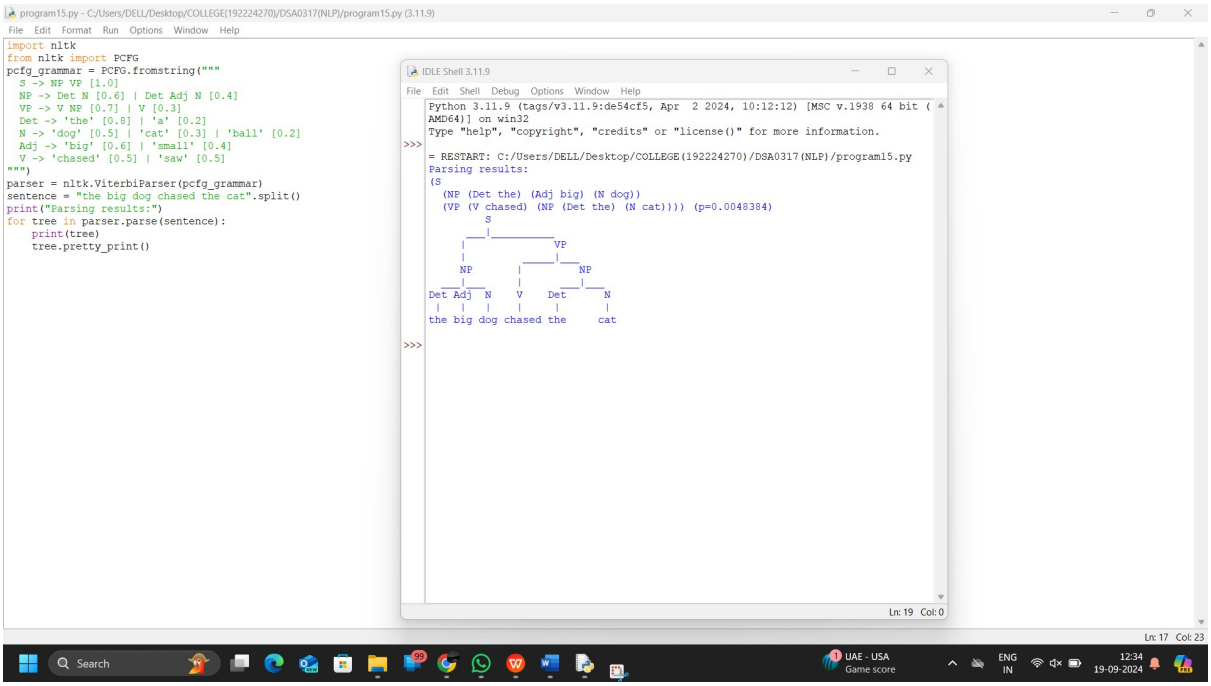
13) Generate a parse tree for a given sentence using a context-free grammar using python program.



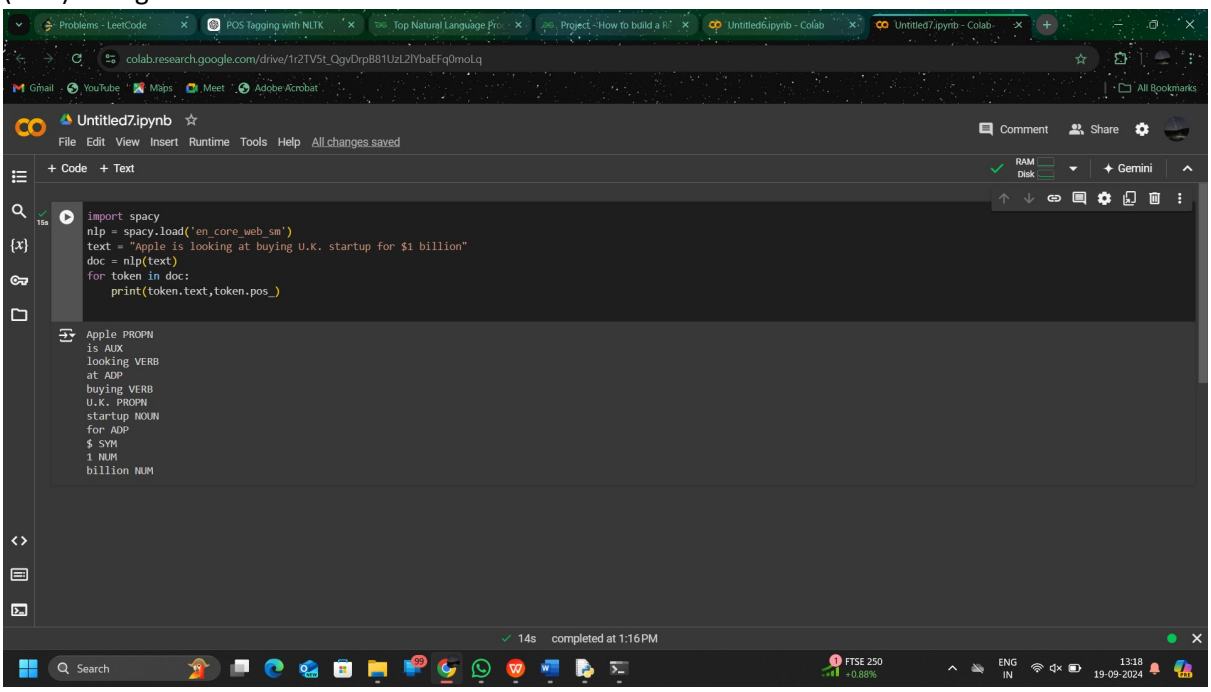
14) Create a program in python to check for agreement in sentences based on a context-free grammar's rules.



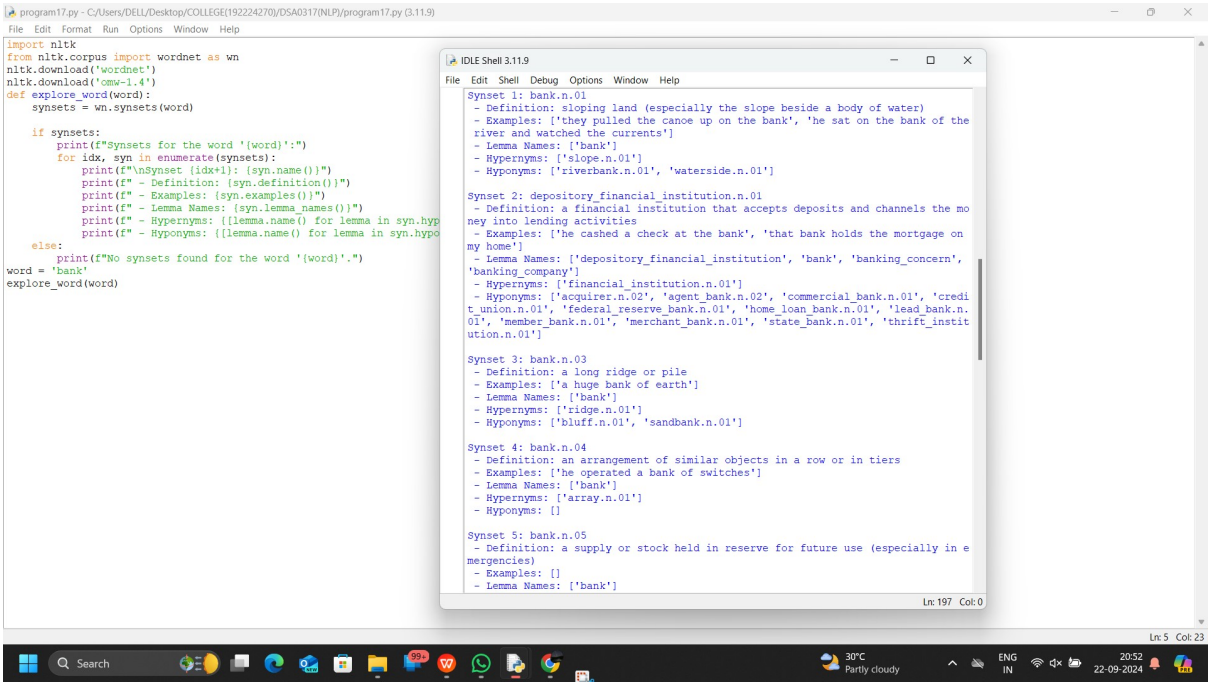
15) Implement probabilistic context-free grammar parsing for a sentence using python.



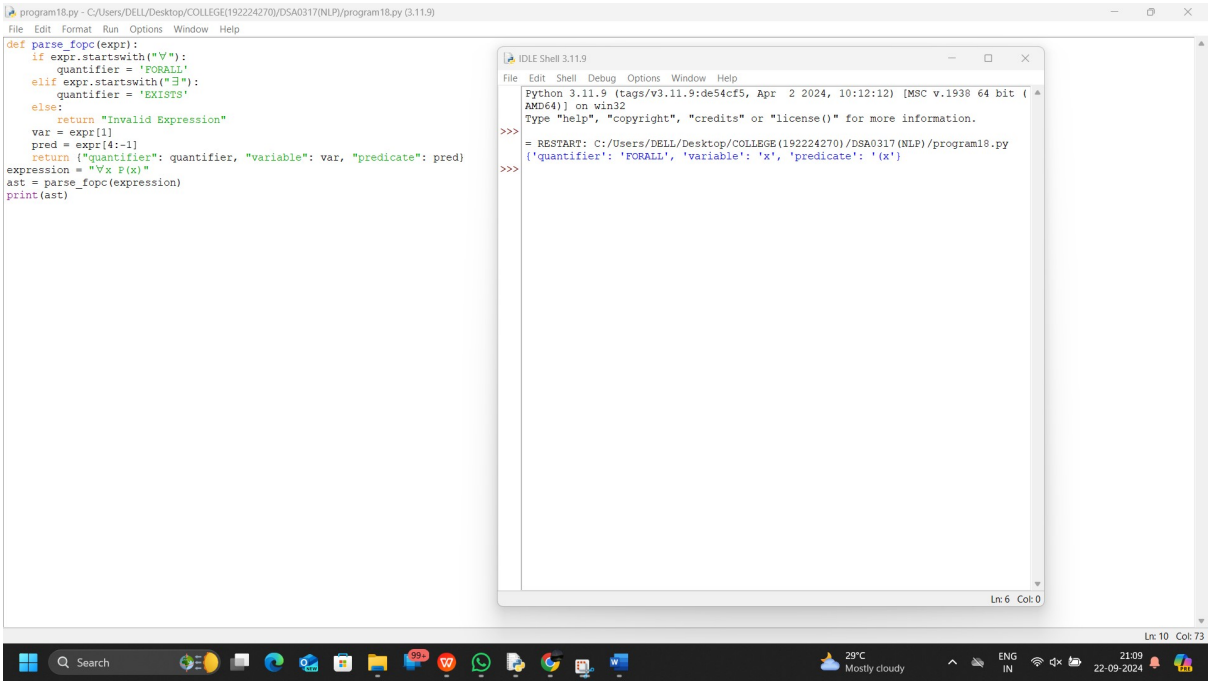
16) Implement a Python program using the SpaCy library to perform Named Entity Recognition (NER) on a given text.



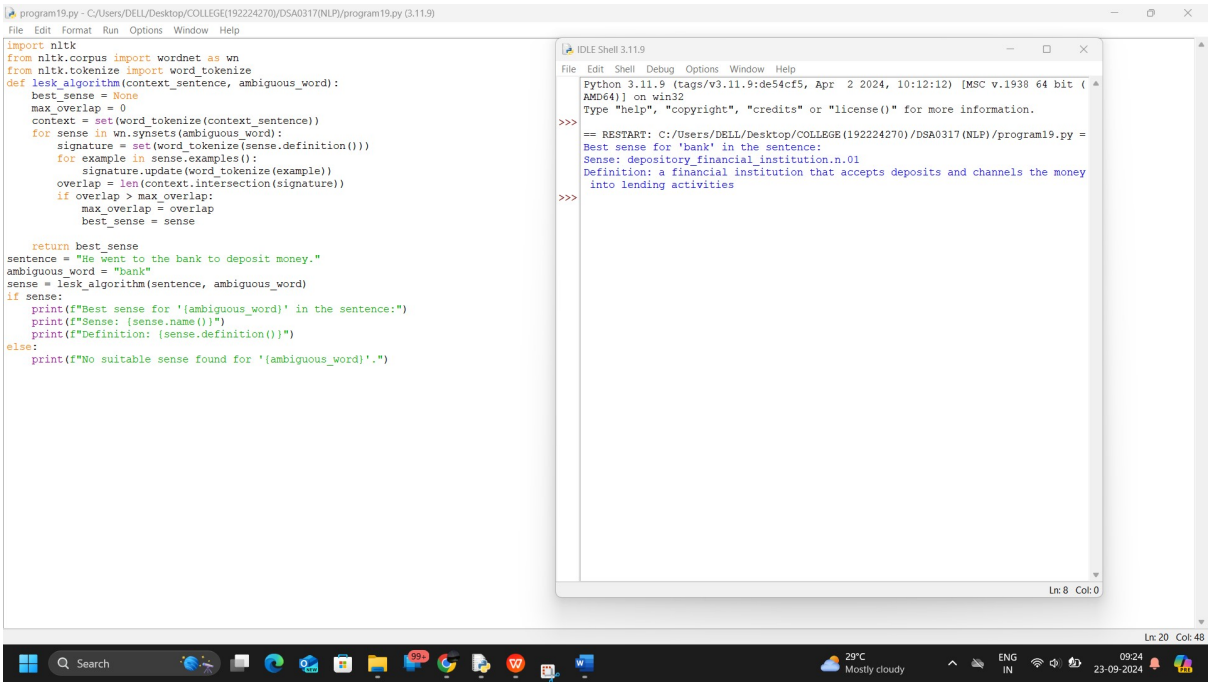
17) Write program demonstrates how to access WordNet, a lexical database, to retrieve synsets and explore word meanings in python.



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



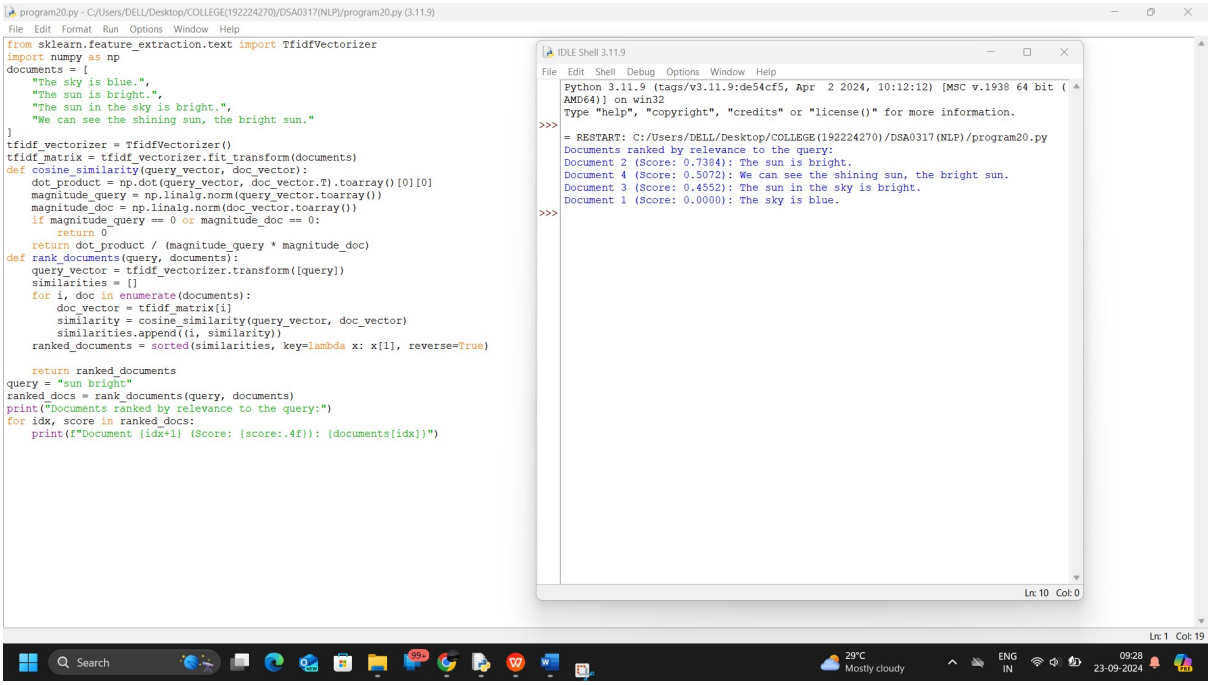
19) Create a program for word sense disambiguation using the Lesk algorithm using python.



```
import nltk
from nltk.corpus import wordnet as wn
from nltk.tokenize import word_tokenize
def lesk_algorithm(context_sentence, ambiguous_word):
    best_sense = None
    max_overlap = 0
    context = set(word_tokenize(context_sentence))
    for sense in wn.synsets(ambiguous_word):
        signature = set(word_tokenize(sense.definition()))
        for example in sense.examples():
            signature.update(word_tokenize(example))
        overlap = len(context.intersection(signature))
        if overlap > max_overlap:
            max_overlap = overlap
            best_sense = sense

    return best_sense
sentence = "He went to the bank to deposit money."
ambiguous_word = "bank"
sense = lesk_algorithm(sentence, ambiguous_word)
if sense:
    print(f"Best sense for '{ambiguous_word}' in the sentence:")
    print(f"Sense: {sense.name()}")
    print(f"Definition: {sense.definition()}")
else:
    print(f"No suitable sense found for '{ambiguous_word}'.")
```

```
Python 3.11.9 (tags/v3.11.9:de54cf5, Apr  2 2024, 10:12:12) [MSC v.1938 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
== RESTART: C:/Users/DELL/Desktop/COLLEGE(192224270)/DSA0317(NLP)/program19.py =
Best sense for 'bank' in the sentence:
Sense: depository_financial_institution.n.01
Definition: a financial institution that accepts deposits and channels the money
 into lending activities
>>>
```

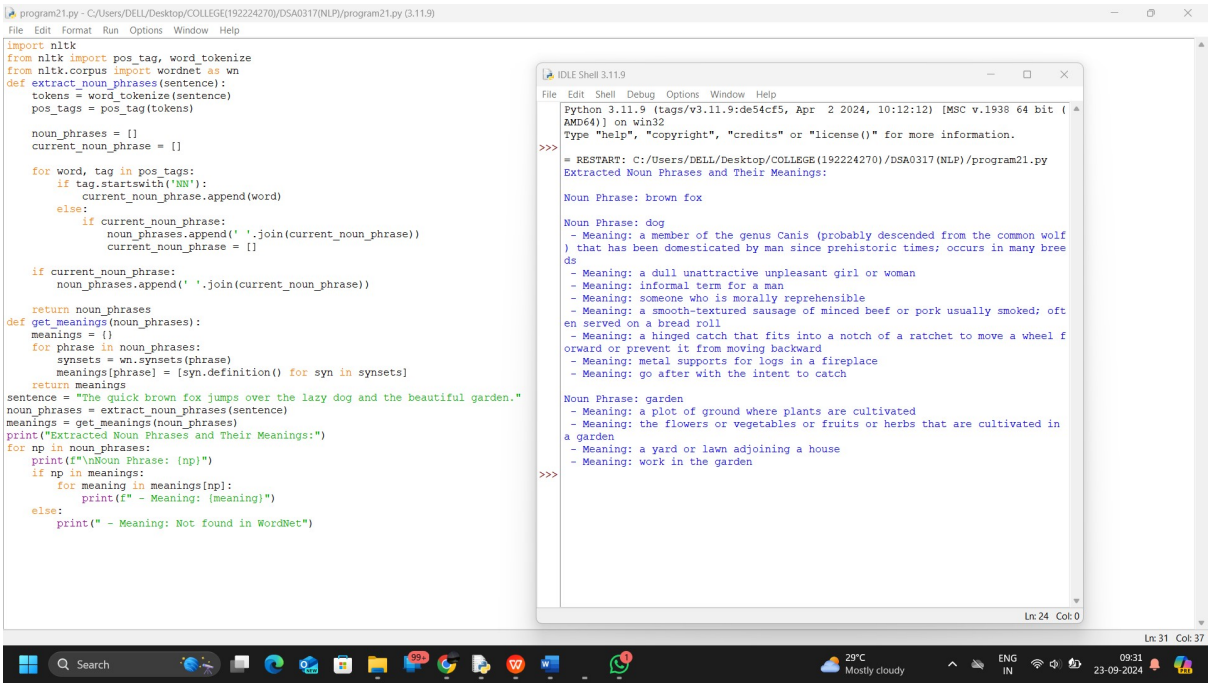
20) Implement a basic information retrieval system using TF-IDF (Term Frequency-Inverse Document Frequency) for document ranking using python.



```
from sklearn.feature_extraction.text import TfidfVectorizer
import numpy as np
documents = [
    "The sky is blue.",
    "The sun is bright.",
    "The sun in the sky is bright.",
    "We can see the shining sun, the bright sun."
]
tfidf_vectorizer = TfidfVectorizer()
tfidf_matrix = tfidf_vectorizer.fit_transform(documents)
def cosine_similarity(query_vector, doc_vector):
    dot_product = np.dot(query_vector, doc_vector.T).toarray()[0][0]
    magnitude_query = np.linalg.norm(query_vector.toarray())
    magnitude_doc = np.linalg.norm(doc_vector.toarray())
    if magnitude_query == 0 or magnitude_doc == 0:
        return 0
    return dot_product / (magnitude_query * magnitude_doc)
def rank_documents(query, documents):
    query_vector = tfidf_vectorizer.transform([query])
    similarities = []
    for i, doc in enumerate(documents):
        doc_vector = tfidf_matrix[i]
        similarity = cosine_similarity(query_vector, doc_vector)
        similarities.append((i, similarity))
    ranked_documents = sorted(similarities, key=lambda x: x[1], reverse=True)

    return ranked_documents
query = "sun bright"
ranked_docs = rank_documents(query, documents)
print("Documents ranked by relevance to the query:")
for idx, score in ranked_docs:
    print(f"Document {idx+1} (Score: {score:.4f}): {documents[idx]}")
```

```
Python 3.11.9 (tags/v3.11.9:de54cf5, Apr  2 2024, 10:12:12) [MSC v.1938 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/DELL/Desktop/COLLEGE(192224270)/DSA0317(NLP)/program20.py
Documents ranked by relevance to the query:
Document 2 (Score: 0.7384): The sun is bright.
Document 4 (Score: 0.5072): We can see the shining sun, the bright sun.
Document 3 (Score: 0.4552): The sun in the sky is bright.
Document 1 (Score: 0.0000): The sky is blue.
>>>
```

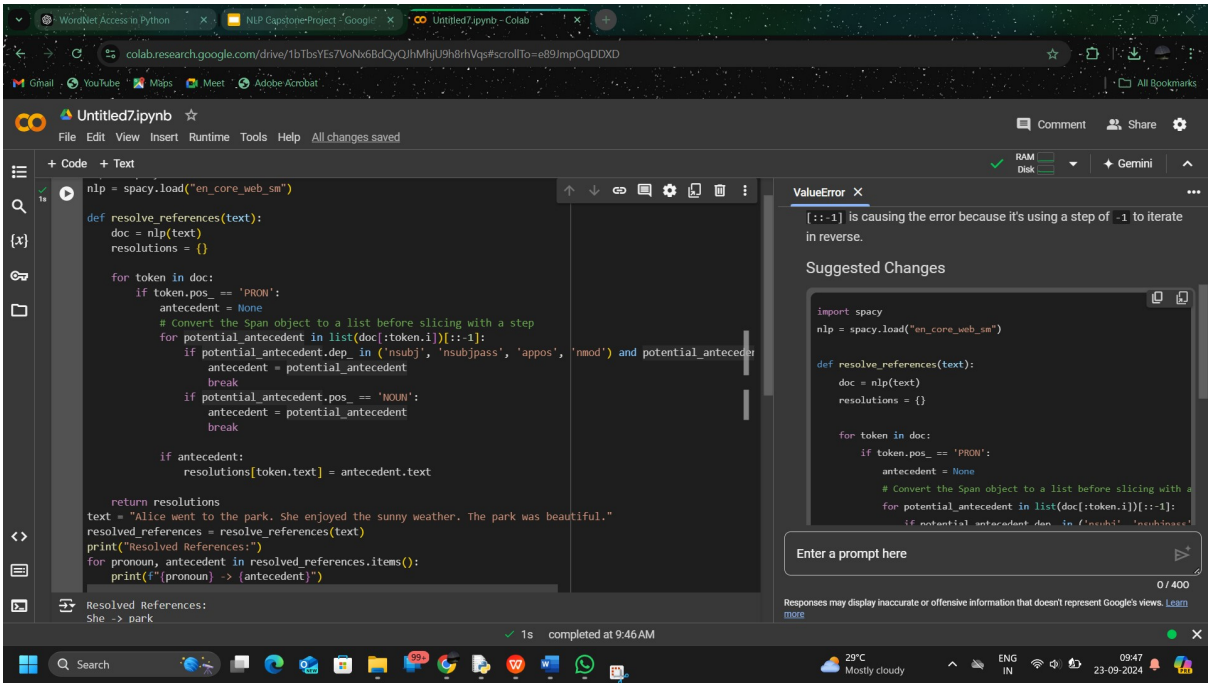
21) Create a python program that performs syntax-driven semantic analysis by extracting noun phrases and their meanings from a sentence.



```
import nltk
from nltk import pos_tag, word_tokenize
from nltk.corpus import wordnet as wn
def extract_noun_phrases(sentence):
    tokens = word_tokenize(sentence)
    pos_tags = pos_tag(tokens)

    noun_phrases = []
    current_noun_phrase = []

    for word, tag in pos_tags:
        if tag.startswith('NN'):
            current_noun_phrase.append(word)
        else:
            if current_noun_phrase:
                noun_phrases.append(' '.join(current_noun_phrase))
                current_noun_phrase = []

    if current_noun_phrase:
        noun_phrases.append(' '.join(current_noun_phrase))

    return noun_phrases
def get_meanings(noun_phrases):
    meanings = {}
    for phrase in noun_phrases:
        synsets = wn.synsets(phrase)
        meanings[phrase] = [syn.definition() for syn in synsets]
    return meanings
sentence = "The quick brown fox jumps over the lazy dog and the beautiful garden."
noun_phrases = extract_noun_phrases(sentence)
meanings = get_meanings(noun_phrases)
print("Extracted Noun Phrases and Their Meanings:")
for np in noun_phrases:
    print(f"\nNoun Phrase: {np}")
    if np in meanings:
        for meaning in meanings[np]:
            print(f" - Meaning: {meaning}")
    else:
        print(" - Meaning: Not found in WordNet")
```

```
Python 3.11.9 (tags/v3.11.9:de54cf5, Apr  2 2024, 10:12:12) [MSC v.1938 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/DELL/Desktop/COLLEGE(192224270)/DSA0317(NLP)/program21.py
Extracted Noun Phrases and Their Meanings:

Noun Phrase: brown fox

Noun Phrase: 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
 - Meaning: a dull unattractive unpleasant girl or woman
 - Meaning: informal term for a man
 - Meaning: someone who is morally reprehensible
 - Meaning: a smooth-textured sausage of minced beef or pork usually smoked; often served on a bread roll
 - Meaning: a hinged catch that fits into a notch of a ratchet to move a wheel forward or prevent it from moving backward
 - Meaning: metal supports for logs in a fireplace
 - Meaning: go after with the intent to catch

Noun Phrase: garden
 - Meaning: a plot of ground where plants are cultivated
 - Meaning: the flowers or vegetables or fruits or herbs that are cultivated in a garden
 - Meaning: a yard or lawn adjoining a house
 - Meaning: work in the garden
>>>
```

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

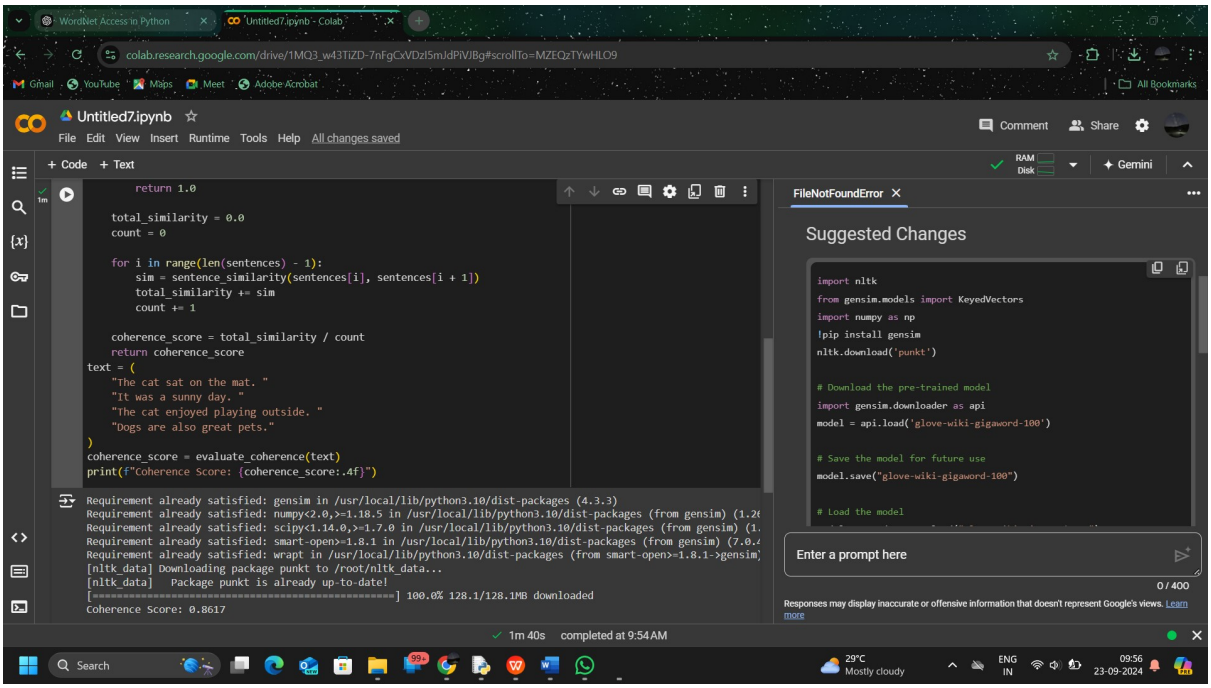


```
nlp = spacy.load("en_core_web_sm")

def resolve_references(text):
    doc = nlp(text)
    resolutions = {}

    for token in doc:
        if token.pos_ == 'PRON':
            antecedent = None
            # Convert the Span object to a list before slicing with a step
            for potential_antecedent in list(doc[:token.i])[::-1]:
                if potential_antecedent.dep_ in ('nsubj', 'nsubjpass', 'appos', 'nmod') and potential_antecedent
                    antecedent = potential_antecedent
                    break
                if potential_antecedent.pos_ == 'NOUN':
                    antecedent = potential_antecedent
                    break

            if antecedent:
                resolutions[token.text] = antecedent.text

    return resolutions
text = "Alice went to the park. She enjoyed the sunny weather. The park was beautiful."
resolved_references = resolve_references(text)
print("Resolved References:")
for pronoun, antecedent in resolved_references.items():
    print(f"{pronoun} -> {antecedent}")
```

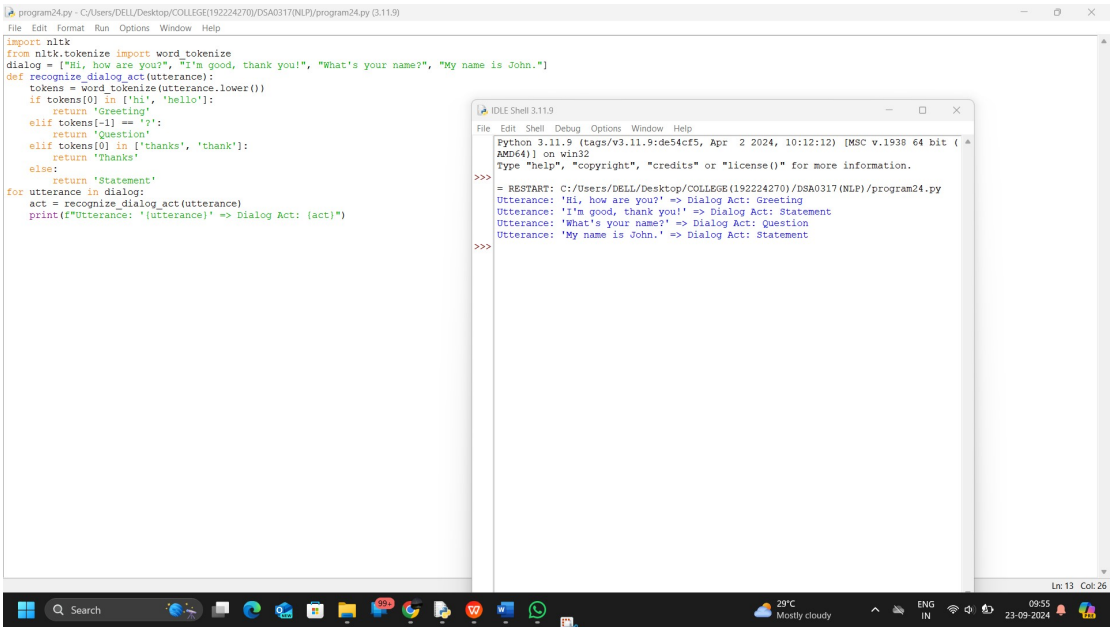
```
Resolved References:
She -> park
```

```
[::-1] is causing the error because it's using a step of -1 to iterate in reverse.

Suggested Changes

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

def resolve_references(text):
    doc = nlp(text)
    resolutions = {}

    for token in doc:
        if token.pos_ == 'PRON':
            antecedent = None
            # Convert the Span object to a list before slicing with a
            for potential_antecedent in list(doc[:token.i])[::-1]:

Enter a prompt here
0 / 400
Responses may display inaccurate or offensive information that doesn't represent Google's views. Learn more
```

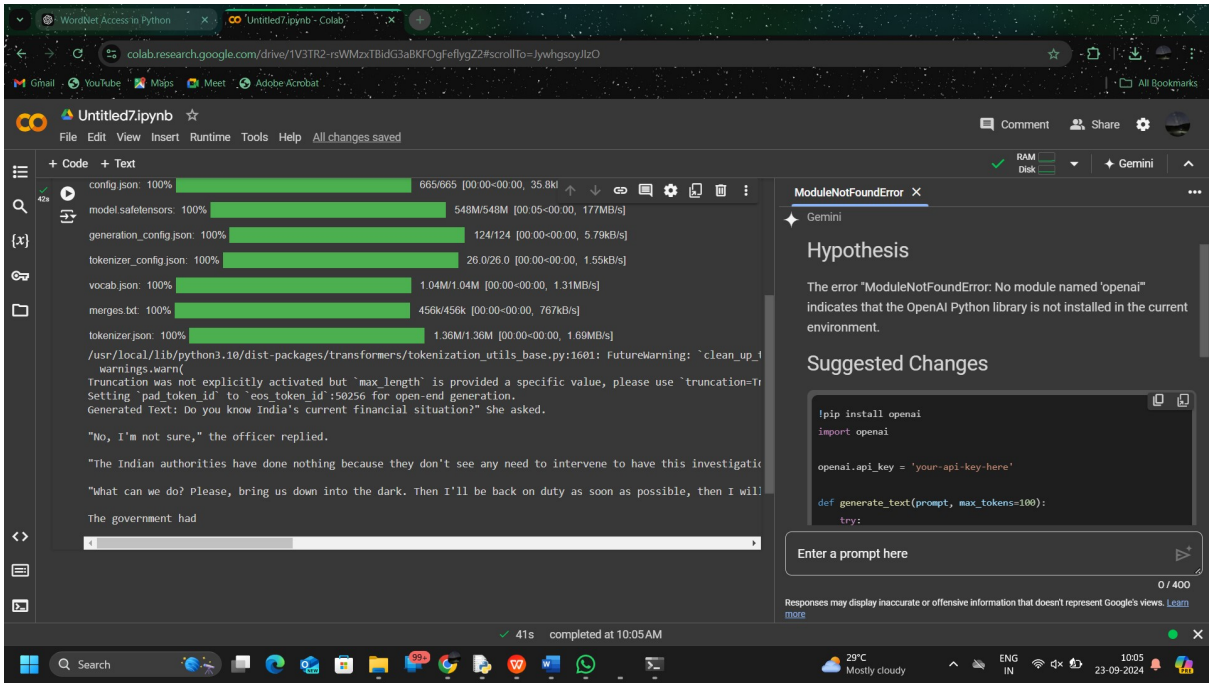
23) Develop a python program that evaluates the coherence of a given text.



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



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.



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