

AARON PULIKKOTTIL(192224270)

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

The screenshot shows a Windows desktop environment. In the foreground, there is an IDLE window titled "program1.py - C:/Users/DELL/Desktop/COLLEGE(192224270)/DSA0317(NLP)/program1.py (3.11.9)". The code in the window is as follows:

```
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)
```

The taskbar at the bottom of the screen shows various application icons, and the system tray indicates it's 12:42, 17-09-2024, with a battery level of 35% and a weather of sunny.

- 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.

The screenshot shows a Windows desktop environment. In the foreground, there is an IDLE window titled "program2.py - C:/Users/DELL/Desktop/COLLEGE(192224270)/DSA0317(NLP)/program2.py (3.11.9)" and an IDLE Shell window titled "IDLE Shell 3.11.9".

The code in the "program2.py" file is:

```
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", "abc", "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'
```

The IDLE Shell window shows the output of running the script:

```
>>> = RESTART: C:/Users/DELL/Desktop/COLLEGE(192224270)/DSA0317(NLP)/program2.py
Python 3.11.9 (tags/v3.11.9:de54cf5, Apr 2 2024, 10:12:12) [MSC v.1938 64 bit (AMD64)]
Type "help", "copyright", "credits" or "license()" for more information.
'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').
```

The taskbar at the bottom of the screen shows various application icons, and the system tray indicates it's 12:00, 17-09-2024, with a battery level of 35% and a weather of NIFTY +0.16%.

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

```

program3.py - C:/Users/DELL/Desktop/COLLEGE(192224270)/DSA0317(NLP)/program3.py (3.11.9)
File Edit Format Run Options Window Help
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)

```

```

IDLE Shell 3.11.9
File Edit Shell Debug Options Window Help
Python 3.11.9 (tags/v3.11.9:9de54cf5, 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 tokenizerspunkt.zip.
[nltk data] Downloading package wordnet to
[nltk data]   C:\Users\DELL\AppData\Roaming\nltk_data...
[nltk data] Unzipping wordnet.zip.
[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', 'feet', 'for', 'best', '.']
Stemmed Words: ['the', 'stripe', 'bat', 'are', 'hang', 'on', 'their', 'feet', '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.

```

program4.py - C:/Users/DELL/Desktop/COLLEGE(192224270)/DSA0317(NLP)/program4.py (3.11.9)
File Edit Format Run Options Window Help
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}")

```

```

IDLE Shell 3.11.9
File Edit Shell Debug Options Window Help
Python 3.11.9 (tags/v3.11.9:9de54cf5, 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.

The screenshot shows a Windows desktop environment. In the top left, there's a file explorer window titled "program5.py - C:/Users/DELL/Desktop/COLLEGE(192224270)/DSA0317(NLP)/program5.py (3.11.9)". Below it is a code editor window with Python code for performing stemming on a list of words using the PorterStemmer library from NLTK. To the right of the code editor is an IDLE Shell window showing the execution of the script. The shell output shows the original words and their stemmed versions. The system tray at the bottom right indicates the date as 17-09-2024 and the time as 13:16.

```

import nltk
from nltk.stem import PorterStemmer
ps = PorterStemmer()
words = ["running", "jumps", "easily", "flying", "cats", "caresses", "happily", "studies"]
stemmed_words = [ps.stem(word) for word in words]
print("Original Words:", words)
print("Stemmed Words:", stemmed_words)

```

```

File Edit Shell Debug Options Window Help
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)/program5.py
Original Words: ['running', 'jumps', 'easily', 'flying', 'cats', 'caresses', 'happily', 'studies']
Stemmed Words: ['run', 'jump', 'easili', 'fli', 'cat', 'caress', 'happili', 'stu di']
>>>

```

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

The screenshot shows a Windows desktop environment. In the top left, there's a file explorer window titled "program6.py - C:/Users/DELL/Desktop/COLLEGE(192224270)/DSA0317(NLP)/program6.py (3.11.9)". Below it is a code editor window with Python code for generating text using a bigram model. To the right of the code editor is an IDLE Shell window showing the execution of the script. The shell output shows the generated text starting with "I love to eat pasta too . Pizza is my". The system tray at the bottom right indicates the date as 17-09-2024 and the time as 13:20.

```

import random
import nltk
from nltk import bigrams, word_tokenize
from collections import defaultdict
text = "I love to eat pizza. I love to eat pasta too. I also enjoy reading books. Pizza is my favor
tokens = word_tokenize(text)
bigram_model = list(bigrams(tokens))
bigram_dict = defaultdict(list)
for w1, w2 in bigram_model:
    bigram_dict[w1].append(w2)
def generate_bigram_text(start_word, num_words):
    current_word = start_word
    generated_words = [current_word]
    for _ in range(num_words - 1):
        next_words = bigram_dict.get(current_word)
        if not next_words:
            break
        current_word = random.choice(next_words)
        generated_words.append(current_word)
    return ' '.join(generated_words)
start_word = "I"
generated_text = generate_bigram_text(start_word, 10)
print("Generated Text:", generated_text)

```

```

File Edit Shell Debug Options Window Help
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)/program6.py
Generated Text: I love to eat pasta too . Pizza is my
>>>

```

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

```

import nltk
from nltk import word_tokenize
from nltk.corpus import stopwords
from nltk.tag import pos_tag
text = "Natural Language Processing is an interesting field of artificial intelligence."
tokens = word_tokenize(text)
pos_tags = pos_tag(tokens)
print("POS Tags:", pos_tags)

```

```

File Edit Format Run Options Window Help
File Edit Shell Debug Options Window Help
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)/program7.py ==
POS Tags: [('Natural', 'JJ'), ('Language', 'NNP'), ('Processing', 'NNP'), ('is', 'VBZ'), ('an', 'DET'), ('interesting', 'JJ'), ('field', 'NN'), ('of', 'IN'), ('artificial', 'JJ'), ('intelligence', 'NN'), ('.', '.')]
>>>

```

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

```

import nltk
from nltk.corpus import brown
from nltk.tag import UnigramTagger, BigramTagger
from nltk.corpus import treebank
nltk.download('brown')
nltk.download('universal_tagset')
nltk.download('treebank')
train_sents = brown.tagged_sents(tagset='universal')
unigram_tagger = UnigramTagger(train_sents)
bigram_tagger = BigramTagger(train_sents, backoff=unigram_tagger)
text = "Natural Language Processing is an interesting field of artificial intelligence."
tokens = nltk.word_tokenize(text)
unigram_tags = unigram_tagger.tag(tokens)
bigram_tags = bigram_tagger.tag(tokens)
print("Unigram Tags:", unigram_tags)
print("Bigram Tags:", bigram_tags)

```

```

File Edit Format Run Options Window Help
File Edit Shell Debug Options Window Help
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)/program8.py ==
[nltk_data] Downloading package brown to
[nltk_data]   C:/Users/DELL/AppData/Roaming/nltk_data...
[nltk_data] Package brown is already up-to-date!
[nltk_data] Downloading package universal_tagset to
[nltk_data]   C:/Users/DELL/AppData/Roaming/nltk_data...
[nltk_data] Package universal_tagset is already up-to-date!
[nltk_data] Downloading package treebank to
[nltk_data]   C:/Users/DELL/AppData/Roaming/nltk_data...
[nltk_data] Package treebank is already up-to-date!
Unigram Tags: [('Natural', 'ADJ'), ('Language', 'NOUN'), ('Processing', 'VERB'),
               ('is', 'VB'), ('an', 'DET'), ('interesting', 'ADJ'), ('field', 'NOUN'), ('of',
               'ADP'), ('artificial', 'ADJ'), ('intelligence', 'NOUN'), ('.', '.')]
Bigram Tags: [('Natural', 'ADJ'), ('Language', 'NOUN'), ('Processing', 'VERB'),
               ('is', 'VERB'), ('an', 'DET'), ('interesting', 'ADJ'), ('field', 'NOUN'), ('of',
               'ADP'), ('artificial', 'ADJ'), ('intelligence', 'NOUN'), ('.', '.')]
>>>

```

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

The screenshot shows a Windows desktop environment. At the top, there is a taskbar with various icons. Below it, two windows are open: a code editor window titled "program9.py" and a Python IDLE Shell window titled "IDLE Shell 3.11.9".

Code Editor (program9.py):

```
program9.py - C:/Users/DELL/Desktop/COLLEGE(192224270)/DSA0317(NLP)/program9.py (3.11.9)
File Edit Format Run Options Window Help
import nltk
from nltk import word_tokenize
from nltk.tag import RegexpTagger
text = "Natural Language Processing is an interesting field of artificial intelligence."
tokens = word_tokenize(text)
patterns = [
    ('.*ing$', 'VBG'), # Gerunds
    ('.*ed$', 'VBD'), # Past tense verbs
    ('.*es$', 'VBZ'), # 3rd person singular present
    ('.*\'ss$', 'VBN'), # Possessive nouns
    ('.*ss$', 'NNS'), # Plural nouns
    ('t.*[a-z]*s$', 'NNP'), # Proper nouns
    ('t.*', 'NN') # Default rule
]
regexp_tagger = RegexpTagger(patterns)
tagged_tokens = regexp_tagger.tag(tokens)
print("Tagged Tokens:", tagged_tokens)
```

IDLE Shell (IDLE Shell 3.11.9):

```
File Edit Shell Debug Options Window Help
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)/program9.py
Tagged Tokens: [('Natural', 'NNP'), ('Language', 'NNP'), ('Processing', 'VBG'), ('is', 'NN'), ('an', 'NN'), ('interesting', 'VBN'), ('field', 'NN'), ('of', 'NN'), ('artificial', 'NN'), ('intelligence', 'NN'), ('.', 'NN')]
>>>
```

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

The screenshot shows a Windows desktop environment. At the top, there is a taskbar with various icons. Below it, two windows are open: a code editor window titled "program10.py" and a Python IDLE Shell window titled "IDLE Shell 3.11.9".

Code Editor (program10.py):

```
program10.py - C:/Users/DELL/Desktop/COLLEGE(192224270)/DSA0317(NLP)/program10.py (3.11.9)
File Edit Format Run Options Window Help
import nltk
from nltk.corpus import brown
from nltk.tag import UnigramTagger
from nltk import word_tokenize
train_sents = brown.tagged_sents(tagset='universal')
unigram_tagger = UnigramTagger(train_sents)
text = "He is going to the store to buy some groceries."
tokens = word_tokenize(text)
initial_tags = unigram_tagger.tag(tokens)
print("Initial Tags: ", initial_tags)
def apply_transformation_rules(tagged_sentence):
    transformed_sentence = []
    for word, tag in tagged_sentence:
        for word, tag in tagged_sentence:
            if tag == 'NOUN' and word.endswith('ing'):
                transformed_sentence.append((word, 'VERB'))
            else:
                transformed_sentence.append((word, tag))
    return transformed_sentence
transformed_tags = apply_transformation_rules(initial_tags)
print("Transformed Tags: ", transformed_tags)
```

IDLE Shell (IDLE Shell 3.11.9):

```
File Edit Shell Debug Options Window Help
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)/program10.py
Initial Tags: [('He', 'PRON'), ('is', 'VERB'), ('going', 'VERB'), ('to', 'PRT'), ('buy', 'VERB'), ('some', 'DET'), ('the', 'DET'), ('store', 'NOUN'), ('to', 'PRT'), ('buy', 'VERB'), ('some', 'DET'), ('groceries', 'NOUN'), ('.', '.')]
Transformed Tags: [('He', 'PRON'), ('is', 'VERB'), ('going', 'VERB'), ('to', 'PRT'), ('buy', 'VERB'), ('buy', 'VERB'), ('some', 'DET'), ('groceries', 'NOUN'), ('.', '.')]
```

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

The screenshot shows a Windows desktop environment. In the foreground, there is a Python script named `program11.py` and an IDLE Shell window. The script defines a context-free grammar and uses a RecursiveDescentParser to parse the sentence "the big dog saw a cat". The IDLE shell displays the resulting parse tree:

```

>>> = RESTART: C:/Users/DELL/Desktop/COLLEGE(192224270)/DSA0317(NLP)/program11.py
Parsing results:
(S
  (NP (Det the) (Adj big) (N dog))
  (VP (V saw) (NP (Det a) (N cat))))
)
S
|
NP
|
Det Adj N
|   |   |
the big dog saw a cat
  
```

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

The screenshot shows a Windows desktop environment. In the foreground, there is a Python script named `program12.py` and an IDLE Shell window. The script defines a context-free grammar and uses an EarleyChartParser to parse the sentence "John saw the big dog". The IDLE shell displays the resulting parse tree:

```

>>> == RESTART: C:/Users/DELL/Desktop/COLLEGE(192224270)/DSA0317(NLP)/program12.py
Parsing results:
(S (NP John) (VP (V saw) (NP (Det the) (Adj big) (N dog))))
)
S
|
NP
|
NP V Det Adj N
|   |   |   |
John saw the big dog
  
```

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

```

program13.py - C:/Users/DELL/Desktop/COLLEGE(192224270)/DSA0317(NLP)/program13.py (3.11.9)
File Edit Format Run Options Window Help
import nltk
from nltk import CFG
grammar = CFG.fromstring("""
S -> NP VP
NP -> Det N | Det Adj N | 'John'
VP -> V NP | V
Det -> 'the' | 'a'
N -> 'dog' | 'cat' | 'park' | 'ball'
Adj -> 'big' | 'small'
V -> 'saw' | 'chased' | 'ran'
""")
parser = nltk.ChartParser(grammar)
sentence = "John saw the big dog".split()
print("Parse tree(s) for the sentence:")
for tree in parser.parse(sentence):
    print(tree)
    tree.pretty_print()

IDLE Shell 3.11.9
File Edit Shell Debug Options Window Help
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)/program13.py
Parse tree(s) for the sentence:
(S (NP John) (VP (V saw) (NP (Det the) (Adj big) (N dog))))
S
|
VP
|
NP
|
NP   V   Det   Adj   N
|   |   |   |   |
John saw the   big dog
>>>

```

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

```

program14.py - C:/Users/DELL/Desktop/COLLEGE(192224270)/DSA0317(NLP)/program14.py (3.11.9)
File Edit Format Run Options Window Help
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 NP_SG | V_SG
VP_PL -> V_PL NP_PL | V_PL
Det_SG -> 'the' | 'a'
Det_PL -> 'the'
N_SG -> 'dog' | 'cat'
N_PL -> 'dogs' | 'cats'
V_SG -> 'chases' | 'runs'
V_PL -> 'chase' | 'run'
NP -> NP_SG | NP_PL
""")
parser = nltk.ChartParser(grammar)
def check_agreement(sentence):
    tokens = sentence.split()
    try:
        parse_trees = list(parser.parse(tokens))
        if parse_trees:
            print(f"'{sentence}' is grammatically correct!")
            for tree in parse_trees:
                tree.pretty_print()
        else:
            print(f"'{sentence}' is not grammatically correct.")
    except ValueError:
        print(f"'{sentence}' is not grammatically correct.")
sentences = [
    "the dog chases the cat",
    "the dogs chase the cat",
    "the dog chase the cat",
    "the dogs chases the cat"
]
for sentence in sentences:
    check_agreement(sentence)

IDLE Shell 3.11.9
File Edit Shell Debug Options Window Help
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)/program14.py
'the dog chases the cat' is grammatically correct!
S
|
VP_SG
|
NP_SG   NP_SG
|       |
Det_SG   N_SG   Det_SG   N_SG
|       |       |       |
the   dog   chases   the   cat
'the dogs chase the cat' is grammatically correct!
S
|
VP_PL
|
NP
|
NP_PL   NP   NP_SG
|       |   |
Det_PL   N_PL   V_PL   Det_SG   N_SG
|       |       |       |
the   dogs   chase   the   cat
'the dog chase the cat' is not grammatically correct.
'the dogs chases the cat' is not grammatically correct.
>>>

```

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

The screenshot shows a Python IDLE Shell window running Python 3.11.9. The code defines a PCFG grammar and parses the sentence "the big dog chased the cat". The output shows the probability of 0.0048384 and a parse tree diagram.

```
program15.py - C:/Users/DELL/Desktop/COLLEGE(192224270)/DSA0317(NLP)/program15.py (3.11.9)
File Edit Shell Debug Options Window Help
import nltk
from nltk import PCFG
pcfg_grammar = PCFG.fromstring("""
S -> NP VP [1.0]
NP -> Det N [0.6] | Det Adj N [0.4]
VP -> V NP [0.7] | V [0.3]
Det -> 'the' [0.8] | 'a' [0.2]
N -> 'dog' [0.5] | 'cat' [0.3] | 'ball' [0.2]
Adj -> 'big' [0.6] | 'small' [0.4]
V -> 'chased' [0.5] | 'saw' [0.5]
""")
parser = nltk.ViterbiParser(pcfg_grammar)
sentence = "the big dog chased the cat".split()
print("Parsin results:")
for tree in parser.parse(sentence):
    print(tree)
    tree.pretty_print()

IDLE Shell 3.11.9
File Edit Shell Debug Options Window Help
Python 3.11.9 (tags/v3.11.9:de54cf5, Apr 2 2024, 10:12:12) [MSC v.1930 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/DELL/Desktop/COLLEGE(192224270)/DSA0317(NLP)/program15.py
Parsin results:
(S
  (NP (Det the) (Adj big) (N dog))
  (VP (V chased) (NP (Det the) (N cat))))
  (p=0.0048384)
  S
  |
  NP
  |
  NP
  |
  Det Adj N V Det N
  the big dog chased the cat
>>>
```

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

The screenshot shows a Google Colab notebook titled "Untitled7.ipynb" running on a GPU. It displays SpaCy NER code and its output for the sentence "Apple is looking at buying U.K. startup for \$1 billion". The output shows tokens and their corresponding entity types.

```
Untitled7.ipynb - Colab
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
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_)

Apple PROPN
is AUX
looking VERB
at ADP
buying VERB
U.K. PROPN
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.

```

program17.py - C:/Users/DELL/Desktop/COLLEGE(192224270)/DSA0317(NLP)/program17.py (3.11.9)
File Edit Format Run Options Window Help
import nltk
from nltk.corpus import wordnet as wn
nltk.download('wordnet')
nltk.download('cmo-1.4')
def explore_word(word):
    synsets = wn.synsets(word)

    if synsets:
        print(f"Synsets for the word '{word}':")
        for idx, syn in enumerate(synsets):
            print(f"\t{idx+1}: {syn.name()}")
            print(f"\t\tDefinition: {syn.definition()}")
            print(f"\t\tExamples: {syn.examples()}")
            print(f"\t\tLemma Names: {syn.lemma_names()}")
            print(f"\t\tHypernyms: {[lemma.name() for lemma in syn.hypernyms()]})
            print(f"\t\tHyponyms: {[lemma.name() for lemma in syn.hyponyms()]})")
    else:
        print(f"No synsets found for the word '{word}'.")

word = "bank"
explore_word(word)

```

IDLE Shell 3.11.9

```

File Edit Shell Debug Options Window Help
Synset 1: bank.n.01
- Definition: sloping land (especially the slope beside a body of water)
- Examples: ['they pulled the canoe up on the bank', 'he sat on the bank of the river and watched the currents']
- Lemma Names: ['bank']
- Hypernyms: ['slope.n.01']
- Hyponyms: ['riverbank.n.01', 'waterside.n.01']

Synset 2: depository_financial_institution.n.01
- Definition: a financial institution that accepts deposits and channels the money into lending activities
- Examples: ['he cashed a check at the bank', 'that bank holds the mortgage on my home']
- Lemma Names: ['depository_financial_institution', 'bank', 'banking_concern', 'banking_company']
- Hypernyms: ['financial_institution.n.01']
- Hyponyms: ['trustee.n.02', 'agent_bank.n.02', 'commercial_bank.n.01', 'credit_union.n.01', 'federal_reserve_bank.n.01', 'home_loan_bank.n.01', 'lead_bank.n.01', 'member_bank.n.01', 'merchant_bank.n.01', 'state_Bank.n.01', 'thrift_institution.n.01']

Synset 3: bank.n.03
- Definition: a long ridge or pile
- Examples: ['a huge bank of earth']
- Lemma Names: ['bank']
- Hypernyms: ['ridge.n.01']
- Hyponyms: ['bluff.n.01', 'sandbank.n.01']

Synset 4: bank.n.04
- Definition: an arrangement of similar objects in a row or in tiers
- Examples: ['I operated a bank of switches']
- Lemma Names: ['bank']
- Hypernyms: ['array.n.01']
- Hyponyms: []

Synset 5: bank.n.05
- Definition: a supply or stock held in reserve for future use (especially in emergencies)
- Examples: []
- Lemma Names: ['bank']

Ln 197 Col:0

```

Windows Taskbar: Search, File Explorer, Edge, Word, Excel, Powerpoint, 99%, Word, WhatsApp, Google Chrome, Microsoft Edge, 30°C Partly cloudy, ENG IN, 20:52, 22-09-2024, system icons.

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

```

program18.py - C:/Users/DELL/Desktop/COLLEGE(192224270)/DSA0317(NLP)/program18.py (3.11.9)
File Edit Format Run Options Window Help
def parse_fopc(expr):
    if expr.startswith("forall"):
        quantifier = "FORALL"
    elif expr.startswith("exists"):
        quantifier = "EXISTS"
    else:
        return "Invalid Expression"
    var = expr[1]
    pred = expr[4:-1]
    return {"quantifier": quantifier, "variable": var, "predicate": pred}
expression = "\forall x P(x)"
ast = parse_fopc(expression)
print(ast)

```

IDLE Shell 3.11.9

```

File Edit Shell Debug Options Window Help
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)/program18.py
{'quantifier': 'FORALL', 'variable': 'x', 'predicate': '(x)'}

Ln 6 Col:0

```

Windows Taskbar: Search, File Explorer, Edge, Word, Excel, Powerpoint, 99%, Word, WhatsApp, Google Chrome, Microsoft Edge, 29°C Mostly cloudy, ENG IN, 21:09, 22-09-2024, system icons.

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

The screenshot shows a Windows desktop environment. At the top, there's a taskbar with various icons. Below it, two windows are open: a code editor window titled "program19.py" and a Python IDLE shell window titled "IDLE Shell 3.11.9".

Code Editor (Left Window):

```

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"\nBest 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}'.")

```

IDLE Shell (Right Window):

```

File Edit Shell Debug Options Window Help
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.

The screenshot shows a Windows desktop environment. At the top, there's a taskbar with various icons. Below it, two windows are open: a code editor window titled "program20.py" and a Python IDLE shell window titled "IDLE Shell 3.11.9".

Code Editor (Left Window):

```

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 idx, doc in enumerate(documents):
        doc_vector = tfidf_matrix[idx]
        similarity = cosine_similarity(query_vector, doc_vector)
        similarities.append((idx, 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]}")

```

IDLE Shell (Right Window):

```

File Edit Shell Debug Options Window Help
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.

```

program21.py - C:/Users/DELL/Desktop/COLLEGE(192224270)/DSA0317(NLP)/program21.py (3.11.9)
File Edit Format Run Options Window Help
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")

```

IDLE Shell 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.

```

WordNet Access in Python
Untitled7.ipynb - Colab
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
ValueError X
[::1] is causing the error because it's using a step of -1 to iterate in reverse.
Suggested Changes
Enter a prompt here
Responses may display inaccurate or offensive information that doesn't represent Google's views. Learn more

```

```

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.pos_ == 'NOUN':
                    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}")

```

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

The screenshot shows a Google Colab notebook titled "Untitled7.ipynb". The code in the cell evaluates the coherence of a given text. It uses a loop to calculate sentence similarity and then divides the total similarity by the count of sentences. The text provided is a short story about a cat. The output shows the coherence score as 0.8617. A sidebar on the right displays suggested changes for installing the gensim library and its dependencies.

```
return 1.0
total_similarity = 0.0
count = 0

for i in range(len(sentences) - 1):
    sim = sentence_similarity(sentences[i], sentences[i + 1])
    total_similarity += sim
    count += 1

coherence_score = total_similarity / count
return coherence_score

text = (
    "The cat sat on the mat."
    "It was a sunny day."
    "The cat enjoyed playing outside."
    "Dogs are also great pets."
)
coherence_score = evaluate_coherence(text)
print(f"Coherence Score: {coherence_score:.4f}")
```

Requirement already satisfied: gensim in /usr/local/lib/python3.10/dist-packages (4.3.3)
Requirement already satisfied: numpy<2.0,>-1.18.5 in /usr/local/lib/python3.10/dist-packages (from gensim) (1.26.3)
Requirement already satisfied: scipy<1.14.0,>-1.7.0 in /usr/local/lib/python3.10/dist-packages (from gensim) (1.7.0)
Requirement already satisfied: smart-open>1.8.1 in /usr/local/lib/python3.10/dist-packages (from gensim) (7.0.0)
Requirement already satisfied: wrapt in /usr/local/lib/python3.10/dist-packages (from smart-open>1.8.1->gensim)
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data] Package punkt is already up-to-date!
[=====] 100.0% 128.1/128.1MB downloaded
Coherence Score: 0.8617

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

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

The screenshot shows a Windows desktop with two windows open. On the left is a code editor window for "program24.py" containing Python code that tokenizes utterances and identifies dialog acts (Greeting, Question, Statement, Thank). On the right is an IDLE Shell window showing the execution of the code and the resulting dialog act recognition for a sample conversation.

```
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}")
```

```
File Edit Shell Debug Options Window Help
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/COLLEGE192224270/DBSA0317(NLP)/program24.py
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.

```
config.json: 100% 665/665 [00:00<00:00, 35.8kB/s]
model.safetensors: 100% 548M/548M [00:05<00:00, 177MB/s]
generation_config.json: 100% 124/124 [00:00<00:00, 5.79kB/s]
tokenizer_config.json: 100% 26.0/26.0 [00:00<00:00, 1.55kB/s]
vocab.json: 100% 1.04M/1.04M [00:00<00:00, 1.31MB/s]
merges.txt: 100% 458k/458k [00:00<00:00, 767kB/s]
tokenizer.json: 100% 1.36M/1.36M [00:00<00:00, 1.69MB/s]
/usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py:1601: FutureWarning: `clean_up_
warnings.warn(
Truncation was not explicitly activated but `max_length` is provided a specific value, please use `truncation=True` when setting `pad_token_id` to `eos_token_id` for open-end generation.
Generated Text: Do you know India's current financial situation?" She asked.
"No, I'm not sure," the officer replied.
"The Indian authorities have done nothing because they don't see any need to intervene to have this investigation.
"What can we do? Please, bring us down into the dark. Then I'll be back on duty as soon as possible, then I will.
The government had
```

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

```
english_text = "Hello, how are you today?"
french_translation = translate_text(english_text)
print("Translated Text (French):")
print(french_translation)

The secret 'HF_TOKEN' does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
warnings.warn(
tokenizer_config.json: 100% 42.0/42.0 [00:00<00:00, 663kB/s]
source.spm: 100% 778k/778k [00:00<00:00, 6.39MB/s]
target.spm: 100% 802k/802k [00:00<00:00, 8.05MB/s]
vocab.json: 100% 1.34M/1.34M [00:00<00:00, 10.3MB/s]
config.json: 100% 1.42k/1.42k [00:00<00:00, 30.9kB/s]
/usr/local/lib/python3.10/dist-packages/transformers/models/marian/tokenization_marian.py:175: UserWarning: Recommended: pip install sacremoses.
warnings.warn("Recommended: pip install sacremoses.")
/usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py:1601: FutureWarning: `clean_up_tokenization_spaces` was not set. It will be set to `True` by default.
warnings.warn(
pytorch_model.bin: 100% 301M/301M [00:04<00:00, 86.5MB/s]
generation_config.json: 100% 293/293 [00:00<00:00, 3.75kB/s]
Translated Text (French):
Bonjour, comment allez-vous aujourd'hui?
```