Program 1: Text Preprocessing using NLTK

```
import nltk
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer

# (Run these once in your environment, then you can comment them)
# nltk.download('punkt')
# nltk.download('stopwords')
# nltk.download('punkt_tab')

# 1. Input text
text = "Natural Language Processing makes computers understand human
language. It is very interesting!"

# 2. Sentence Tokenization
sentences = sent_tokenize(text)
print("Sentence Tokenization:")
for s in sentences:
    print(" -", s)

# 3. Word Tokenization
words = word_tokenize(text)
print("\nWord Tokenization:")
print(words)

# 4. Stopword Removal
stop_words = set(stopwords.words('english'))
filtered_words = [w for w in words if w.lower() not in stop_words and
w.isalpha()]
print("\nAfter Stopword Removal:")
print(filtered_words)

# 5. Stemming using Porter Stemmer
ps = PorterStemmer()
stemmed_words = [ps.stem(w) for w in filtered_words]
print("\nAfter Stemming (Porter Stemmer):")
print(stemmed_words)
```

Program 2: POS Tagging and Lemmatization using NLTK

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet

# Download if needed:
# nltk.download('punkt')
# nltk.download('averaged_perceptron_tagger')
# nltk.download('wordnet')
# nltk.download('averaged_perceptron_tagger_eng')


# Helper function to convert NLTK POS tags to WordNet POS tags
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  # default to noun

# Input sentence
text = "The striped bats are hanging on their feet for best"

# Tokenization
words = word_tokenize(text)

# POS Tagging
pos_tags = nltk.pos_tag(words)
print("POS Tags:")
print(pos_tags)

# Lemmatization using POS tags
lemmatizer = WordNetLemmatizer()
lemmatized_words = [lemmatizer.lemmatize(w, get_wordnet_pos(t)) for (w, t)
in pos_tags]

print("\nLemmatized Words:")
print(lemmatized_words)
```

Program 3: Chunking and Named Entity Recognition using NLTK

```
# Download if needed:
# nltk.download('punkt')
# nltk.download('averaged_perceptron_tagger')
# nltk.download('wordnet')
# nltk.download('averaged_perceptron_tagger_eng')
# nltk.download('maxent_ne_chunker_tab')
# nltk.download('words')

import nltk
from nltk import pos_tag, ne_chunk
from nltk.tokenize import word_tokenize
from nltk.chunk import RegexpParser

# Input sentence
text = "Barack Obama visited Microsoft headquarters in Washington."

# Tokenization
words = word_tokenize(text)

# POS Tagging
pos_tags = pos_tag(words)
print("POS Tags:")
print(pos_tags)

# Chunking grammar (Noun Phrase)
grammar = "NP: {<DT>?<JJ>*<NN.*>+}"
chunk_parser = RegexpParser(grammar)
chunked_tree = chunk_parser.parse(pos_tags)

print("\nChunked Phrases:")
print(chunked_tree)

# Named Entity Recognition
ner_tree = ne_chunk(pos_tags)
print("\nNamed Entities:")
print(ner_tree)
```

Program 4: TF-IDF Representation and Parsing using CFG (Chart Parser)

```
# ------------ PART A: TF-IDF ----------------
from sklearn.feature_extraction.text import TfidfVectorizer

corpus = [
    "Natural Language Processing is fun",
    "Language Processing and Machine Learning are related",
    "I love learning NLP and Machine Learning"
]

vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(corpus)

print("----- TF-IDF Vocabulary -----")
print(vectorizer.get_feature_names_out())

print("\n----- TF-IDF Matrix -----")
print(X.toarray())


# ------------ PART B: CFG Parsing using Chart Parser ----------------
import nltk
from nltk import CFG
from nltk.parse import ChartParser

grammar_text = """
S -> NP VP
NP -> Det N | Det Adj N | N
VP -> V NP | V
Det -> 'the' | 'a'
N -> 'boy' | 'girl' | 'apple' | 'telescope'
Adj -> 'small' | 'young'
V -> 'eats' | 'sees'
"""

grammar = CFG.fromstring(grammar_text)
parser = ChartParser(grammar)

sentence = "the young boy sees a small apple"
tokens = sentence.split()

print("\n----- Parsing Input Sentence -----")
print(tokens)

print("\n----- Parse Tree(s) -----")
```

```
for tree in parser.parse(tokens):
    print(tree)
    # tree.draw()  # Optional: shows graphical parse tree if running
locally
```

Program 5: N-gram Modeling and Probability Calculation

```
corpus = ["I love NLP", "I love machine learning", "NLP is fun", "I love
learning NLP"]
S = [["<s>"] + s.lower().split() + ["</s>"] for s in corpus]

uni = Counter(w for sent in S for w in sent)
bi  = Counter((sent[i], sent[i+1]) for sent in S for i in range(len(sent)-
1))
V = len(uni)

def bigram_prob(prev, w):
    return (bi[(prev, w)] + 1) / (uni[prev] + V)    # Add-one smoothing

def sent_prob(sentence):
    ws = ["<s>"] + sentence.lower().split() + ["</s>"]
    p = 1.0
    for a, b in zip(ws, ws[1:]):
        p *= bigram_prob(a, b)
    return p

print("Unigram counts:", uni)
print("Bigram counts :", bi)
print("P('I love nlp') =", sent_prob("I love nlp"))
```

Program 6: Advanced Named Entity Recognition using Stanford NER Tool

```
import os
from pathlib import Path
from itertools import groupby
import nltk
from nltk.tag import StanfordNERTagger
from nltk import word_tokenize

nltk.download('punkt', quiet=True)

base = Path('/content/stanford-ner-2018-10-16')
if not base.exists():
    print("Downloading Stanford NER (small one-time)...")
    os.system('wget -q https://nlp.stanford.edu/software/stanford-ner-
2018-10-16.zip -P /content')
    os.system('unzip -q /content/stanford-ner-2018-10-16.zip -d /content')

jar = str(base / 'stanford-ner.jar')
model = str(base / 'classifiers' /
'english.all.3class.distsim.crf.ser.gz')

ner = StanfordNERTagger(model, jar, encoding='utf-8')

text = "Barack Obama was born in Hawaii and worked at Microsoft in the
United States."
tags = ner.tag(word_tokenize(text))
print("Token-wise NER Tags:\n", tags)

entities = [(" ".join(w for w,_ in g), lbl)
            for lbl, g in groupby(tags, key=lambda x: x[1]) if lbl != 'O']

print("\nNamed Entities (Grouped):")
for ent, lbl in entities:
    print(f"{ent} -> {lbl}")
```

Program 7: Corpus Analysis using Term Frequency and TF-IDF

```
from sklearn.feature_extraction.text import CountVectorizer,
TfidfVectorizer
import numpy as np

# 1. Sample corpus
corpus = [
    "Natural Language Processing is fun and exciting",
    "Machine Learning and Natural Language Processing are related fields",
    "I love studying NLP and Machine Learning"
]

# 2. Term Frequency using CountVectorizer
count_vec = CountVectorizer()
X_count = count_vec.fit_transform(corpus)
terms = count_vec.get_feature_names_out()

# Choose first document for ranking
doc_id = 0
tf = X_count.toarray()[doc_id]

print("----- Term Frequency (Document 0) -----")
tf_indices = np.argsort(-tf)   # sort descending
for idx in tf_indices:
    if tf[idx] > 0:
        print(f"{terms[idx]} : {tf[idx]}")

# 3. TF-IDF using TfidfVectorizer
tfidf_vec = TfidfVectorizer()
X_tfidf = tfidf_vec.fit_transform(corpus)
tfidf_terms = tfidf_vec.get_feature_names_out()
tfidf = X_tfidf.toarray()[doc_id]

print("\n----- TF-IDF Scores (Document 0) -----")
tfidf_indices = np.argsort(-tfidf)  # sort descending
for idx in tfidf_indices[:10]:      # top 10 terms
    print(f"{tfidf_terms[idx]} : {tfidf[idx]:.4f}")
```

Program 8: Sentiment Analysis using NLTK VADER

```
import nltk
from nltk.sentiment import SentimentIntensityAnalyzer

# Download VADER lexicon (if not already downloaded)
nltk.download('vader_lexicon')

# Initialize analyzer
sia = SentimentIntensityAnalyzer()

# Sample sentences
sentences = [
    "I love this phone, the camera quality is amazing!",
    "This is the worst experience I have ever had.",
    "The movie was okay, not too good, not too bad.",
    "The battery life is terrible but the screen is great.",
]

for text in sentences:
    scores = sia.polarity_scores(text)
    compound = scores['compound']

    # Decide sentiment based on compound score
    if compound >= 0.05:
        label = "Positive"
    elif compound <= -0.05:
        label = "Negative"
    else:
        label = "Neutral"

    print(f"Text   : {text}")
    print(f"Scores : {scores}")
    print(f"Sentiment: {label}")
    print("-" * 50)
```