Date of Implementation: 19/06/2025
Date of Submission: 20/06/2025

Lab 1: Text Processing and Regular Expression

Question 1: Installing NLTK, NLTK.book and Practice the NLP Environment using the exercises 1 and 2 from the given link. Language Processing and Python (nltk.org)

Program Description: Installing and downloading NLTK package in python. Using the above give reference we practice how to use the package in pyhton and perform certain operation as mentioned in the document.

Program Logic

- Libraries used: NLTK
- Using the NLTK library we implement and practices NLP tasks. Different functions are used to check how words in a text are contextually similar to one another.

In [34]:
```
#Import+ing necessory libraries
import nltk
from nltk import *
from nltk.book import *
import re
```

In [35]:
```
#Question 1
print("Question 1\n")


print(text1)


print(sent1)

print(sent2)


print("\nOccurance of the word monstorous with some context: ")
text1.concordance("monstrous")


print("\nWords that are in similar range of context: ")
text1.similar("monstrous")

print("\nCommon contexts shared by two words: ")
text2.common_contexts(["monstrous", "very"])
```

```
Question 1

<Text: Moby Dick by Herman Melville 1851>
['Call', 'me', 'Ishmael', '.']
['The', 'family', 'of', 'Dashwood', 'had', 'long', 'been', 'settled', 'in', 'Suss
ex', '.']

Occurance of the word monstorous with some context:
Displaying 11 of 11 matches:
ong the former , one was of a most monstrous size . ... This came towards us ,
ON OF THE PSALMS . " Touching that monstrous bulk of the whale or ork we have r
ll over with a heathenish array of monstrous clubs and spears . Some were thick
d as you gazed , and wondered what monstrous cannibal and savage could ever hav
that has survived the flood ; most monstrous and most mountainous ! That Himmal
they might scout at Moby Dick as a monstrous fable , or still worse and more de
th of Radney .'" CHAPTER 55 Of the Monstrous Pictures of Whales . I shall ere l
ing Scenes . In connexion with the monstrous pictures of whales , I am strongly
ere to enter upon those still more monstrous stories of them which are to be fo
ght have been rummaged out of this monstrous cabinet there is no telling . But
of Whale - Bones ; for Whales of a monstrous size are oftentimes cast up dead u

Words that are in similar range of context:
true contemptible christian abundant few part mean careful puzzled
mystifying passing curious loving wise doleful gamesome singular
delightfully perilous fearless

Common contexts shared by two words:
am_glad a_pretty a_lucky is_pretty be_glad
true contemptible christian abundant few part mean careful puzzled
mystifying passing curious loving wise doleful gamesome singular
delightfully perilous fearless

Common contexts shared by two words:
am_glad a_pretty a_lucky is_pretty be_glad
```

Question 2: Text Processing (Basics)

1. Define a string containing a paragraph as the value.
Program Logic: Storing a paragrah into a variable *String str_para*

In [36]:
```
str_para = """Formula One (F1) is the highest class of worldwide
 racing for open-wheel single-seater formula racing cars sanctioned by the Fédér
 Internationale de l'Automobile (FIA). The FIA Formula One World Championship ha
 premier forms of motorsport since its inaugural running in 1950 and is often co
 motorsport. The word formula in the name refers to the set of rules all partici
 A Formula One season consists of a series of races, known as Grands Prix. Grand
 countries and continents on either purpose-built circuits or closed roadsz."""
```

2.Write a program to print the number of total words and total unique words in the paragraph.

Program Description: The first step to preprocessing a given text is to find the length of the paragraph or the number of words and charachters in it. Then we find the number of unique words that are available in the paragraph.

Program Logic

- Split the paragraph into a list containg the words of the paragraph using *.split()*

- To find the length use *len()*
- To get the unique words we convert the list into a set using *set()*. A set can have only unique items in them therfore all the repeating words gets cancelled.
- Use *len()* function to get the length of the unique words.

In [37]:
```
print("\nQ2 b")
str_list = str_para.split()
print(str_list)

print("\nLength of the paragraph: ", len(str_list))
print("\nNumber of unique words: ", len(set(str_list)))


Freq_Words = FreqDist(str_para)
print(Freq_Words.most_common(20))
```

```
Q2 b
['Formula', 'One', '(F1)', 'is', 'the', 'highest', 'class', 'of', 'worldwide', 'r
acing', 'for', 'open-wheel', 'single-seater', 'formula', 'racing', 'cars', 'sanct
ioned', 'by', 'the', 'Fédération', 'Internationale', 'de', "l'Automobile", '(FI
A).', 'The', 'FIA', 'Formula', 'One', 'World', 'Championship', 'has', 'been', 'on
e', 'of', 'the', "world's", 'premier', 'forms', 'of', 'motorsport', 'since', 'it
s', 'inaugural', 'running', 'in', '1950', 'and', 'is', 'often', 'considered', 't
o', 'be', 'the', 'pinnacle', 'of', 'motorsport.', 'The', 'word', 'formula', 'in',
'the', 'name', 'refers', 'to', 'the', 'set', 'of', 'rules', 'all', 'participant',
'cars', 'must', 'follow.', 'A', 'Formula', 'One', 'season', 'consists', 'of',
'a', 'series', 'of', 'races,', 'known', 'as', 'Grands', 'Prix.', 'Grands', 'Pri
x', 'take', 'place', 'in', 'multiple', 'countries', 'and', 'continents', 'on', 'e
ither', 'purpose-built', 'circuits', 'or', 'closed', 'roadsz.']

Length of the paragraph:  103

Number of unique words:  78
[(' ', 107), ('e', 53), ('o', 48), ('r', 43), ('n', 43), ('s', 40), ('i', 36),
('a', 35), ('t', 34), ('l', 26), ('c', 18), ('u', 16), ('d', 16), ('h', 15),
('m', 14), ('f', 14), ('p', 13), ('F', 7), ('w', 7), ('g', 6)]
```

3.Find the frequency of all words and also display the most and least frequent word.

Program Description: As part of preprocessing we find the frequency or the number of times a word occurs in the paragraph. Also, we find the least occuring word.

Program Logic

- First tokenize the paragraph using *word_tokenize()* from the NLTK library
- Using the *FreqDist()* we get the frequency distribution of the words in the paragraph.
- The *FreqDist()* function returns a tuple with word occuring frequently comming first.

In [38]:
```
tokenised_str = word_tokenize(str_para)
Freq_tokenwords = FreqDist(tokenised_str).most_common(20)
print("Tokenized List: ",tokenised_str)
print("\nFrequency of words: ",Freq_tokenwords)

#Most frequent and less frequent word
print("\nMost frequent word is '", Freq_tokenwords[0][0],"' with frequency: ", F
print("\nLeast frequent words: ", [w for w in Freq_tokenwords if w[1]==1])
```

```
Tokenized List:  ['Formula', 'One', '(', 'F1', ')', 'is', 'the', 'highest', 'clas
s', 'of', 'worldwide', 'racing', 'for', 'open-wheel', 'single-seater', 'formula',
'racing', 'cars', 'sanctioned', 'by', 'the', 'Fédération', 'Internationale', 'd
e', "l'Automobile", '(', 'FIA', ')', '.', 'The', 'FIA', 'Formula', 'One', 'Worl
d', 'Championship', 'has', 'been', 'one', 'of', 'the', 'world', "'s", 'premier',
'forms', 'of', 'motorsport', 'since', 'its', 'inaugural', 'running', 'in', '195
0', 'and', 'is', 'often', 'considered', 'to', 'be', 'the', 'pinnacle', 'of', 'mot
orsport', '.', 'The', 'word', 'formula', 'in', 'the', 'name', 'refers', 'to', 'th
e', 'set', 'of', 'rules', 'all', 'participant', 'cars', 'must', 'follow', '.',
'A', 'Formula', 'One', 'season', 'consists', 'of', 'a', 'series', 'of', 'races',
',', 'known', 'as', 'Grands', 'Prix', '.', 'Grands', 'Prix', 'take', 'place', 'i
n', 'multiple', 'countries', 'and', 'continents', 'on', 'either', 'purpose-buil
t', 'circuits', 'or', 'closed', 'roadsz', '.']

Frequency of words:  [('of', 7), ('the', 6), ('.', 5), ('Formula', 3), ('One',
3), ('in', 3), ('(', 2), (')', 2), ('is', 2), ('racing', 2), ('formula', 2), ('ca
rs', 2), ('FIA', 2), ('The', 2), ('motorsport', 2), ('and', 2), ('to', 2), ('Gran
ds', 2), ('Prix', 2), ('F1', 1)]

Most frequent word is ' of ' with frequency:  7

Least frequent words:  [('F1', 1)]
```

4.Find the longest word in the paragraph

Program Description: Our aim is to find the longest word that occurs in the paragraph.

Program Logic

- We use list comphrehension to return a tuple containing word indexes and their respective lengths.
- Using the *max()* fn we find the word with the maximum length.

In [39]:
```
word_lengths = [(tokenised_str.index(w),len(w)) for w in tokenised_str]
longest_word = tokenised_str[max(word_lengths, key = lambda x:x[1])[0]]
print("Longest word: ",longest_word)
```

```
Longest word:  Internationale
```

Question 3: Regular Expression Solve exercise 2.1 and 2.2 from Text book of Speech and Language Processing of Daniel Jurafsky and team

Write regular expressions for the following languages. 1.the set of all alphabetic strings;

Program Description: Our aim is to check whether a given string has only alphabets or not.

Program Logic

- Libraries used: re(Regular Expression)
- We compare the test strings with the re pattern using *re.fullmatch()*

Test Cases

- "Hans"-> True
- "test123"-> False
- "lucky05"-> False

In [40]:
```
pattern = r'^[a-zA-Z]*$'
test_strings = ["Hans", "test123", "ABC", "lucky05", "hi_there", "123", "hey!"]

for s in test_strings:
    if re.fullmatch(pattern, s):
        print(f"{s} -> True")
    else:
        print(f"{s }-> False")
```

```
Hans -> True
test123-> False
ABC -> True
lucky05-> False
hi_there-> False
123-> False
hey!-> False
```

2.The set of all lower case alphabetic strings ending in a b;

Program Description: Our aim is to check whether a given string is ending in b or not.

Program Logic

- Libraries used: re(Regular Expression)
- We compare the test strings with the re pattern using *re.fullmatch()*

Test Cases

- "lob"-> True
- "Job"-> True
- "Live"-> False
- "job123"-> False
- "lucky05"-> False

In [41]:
```
pattern2  = r'[a-zA-z]*b'
test_strings2 = ["Job","Live", "347","job123","lob"]
for s in test_strings2:
    if re.fullmatch(pattern2, s):
        print(f"{s} -> True")
```

```
    else:
        print(f"{s} -> False")
```

```
Job -> True
Live -> False
347 -> False
job123 -> False
lob -> True
```

3.The set of all strings from the alphabet a,b such that each a is immediately preceded by and immediately followed by a b

Program Description: Our aim is to check whether a string has the format: the alphabet a,b such that each a is immediately preceded by and immediately followed by a b.

Program Logic

- Libraries used: re(Regular Expression)
- We compare the test strings with the re pattern using *re.fullmatch()*

Test Cases

- "bbbbbabbbb"-> True
- "bbabbb"-> True
- "ab"-> False
- "abb"-> False
- "baba"-> False

In [42]:
```
pattern3 = 'b*(bab)*b*'
test_strings3 = ["bbbbbabbbb", "ab", "baba","bbabbb","abb"]
for s in test_strings3:
    if re.fullmatch(pattern3, s):
        print(f"{s} -> True")
    else:
        print(f"{s} -> False")
```

```
bbbbbabbbb -> True
ab -> False
baba -> False
bbabbb -> True
abb -> False
```

Write regular expressions for the following languages. By “word”, we mean an alphabetic string separated from other words by whitespace, any relevant punctuation, line breaks, and so forth.

1.the set of all strings with two consecutive repeated words (e.g., “Hum- bert Humbert” and “the the” but not “the bug” or “the big bug”);

Program Description: Find an re pattern that will match words that appear consecutively in a sentence.

Program Logic

- Libraries used: re(Regular Expression)

- We use the *re.search()* fn to search if the pattern exist in the sentence or not

Test Cases

- "My grandma is the the mayor of Kerala."-> True
- "Humbert Humbert speaks delusional stuff"-> True
- "There is a cat on top of the mat."-> False

In [43]:
```
pattern = r'\b(\w+)\b(?:[\s,.!?;:-]+)\b\1\b'

test_sent = "My grandma is the the mayor of Trivandrum."
test_sent2 = "Humbert Humbert speaks delusional stuff."
test_sent3 = "There is a cat on top of the mat."
print(test_sent," ->", True if re.search(pattern, test_sent, re.IGNORECASE) else
print(test_sent2," ->", True if re.search(pattern, test_sent2, re.IGNORECASE) el
print(test_sent3 ,"->", True if re.search(pattern, test_sent2, re.IGNORECASE) el
```

```
My grandma is the the mayor of Trivandrum.  -> True
Humbert Humbert speaks delusional stuff.  -> True
There is a cat on top of the mat. -> True
```

2.All strings that start at the beginning of the line with an integer and that end at the end of the line with a word;

Program Description: Find an re pattern that will help us verify that whether the given sentence starts with a number and ends with a word.

Program Logic

- Libraries used: re(Regular Expression)
- Since we are trying to match a string with the pattern we use *re.fullmatch()* fn to check if the sentence follows the pattern or not.

Test Cases

- "123 is the number of my neighbor."-> True
- "Divya has a haemoglobin level of 12."-> False

In [44]:
```
test_sent1 = "123 is the number of my neighbor."
test_sent2 = "Divya has a haemoglobin level of 12."
match1 = re.fullmatch(r'^\d+.*\b[a-zA-Z]+\b$', test_sent1)
match2 = re.fullmatch(r'^\d+.*\b[a-zA-Z]+\b$', test_sent2)
print(test_sent1, "-> ",True if match1 else False)
print(test_sent2, "-> ",True if match2 else False)
```

```
123 is the number of my neighbor. ->  False
Divya has a haemoglobin level of 12. ->  False
```

3.All strings that have both the word grotto and the word raven in them (but not, e.g., words like grottos that merely contain the word grotto);

Program Description: Find an re pattern that will help us verify whether the given sentence has both the words raven and grotto.

Program Logic

- Libraries used: re(Regular Expression)
- We use *re.search()* fn to search if the words exist in the sentence or not

Test Cases

- "The raven flew to the grotto"-> True
- "The raven flew into grottos"-> False

In [45]:
```
text1 = "The raven flew to the grotto"
text2 = "The raven flew into grottos"
if re.search(r'(?=.*\bgrotto\b)(?=.*\braven\b)', text1, re.IGNORECASE):
    print(text1, "-> ", True)
else:
    print(text1, "-> ", False)


if re.search(r'(?=.*\bgrotto\b)(?=.*\braven\b)', text2, re.IGNORECASE):
    print(text2, "-> ", True)
else:
    print(text2, "-> ", False)
```

```
The raven flew to the grotto ->  True
The raven flew into grottos ->  False
```

4.Write a pattern that places the first word of an English sentence in a register. Deal with punctuation.

Program Description: The goal of the program is to extract the first word of an English sentence and store it in a register. It takes care to ignore leading punctuation and whitespace, such as quotation marks or spaces.

Program Logic

- Libraries used: re(Regular Expression)
- The code uses a regular expression (re.match) to locate the first word that begins with a capital letter—as expected in grammatically correct English sentences.
- If a match is found, match.group(1) retrieves the first word (without quotes/punctuation).

Test Cases

- ""Hello there!" he said."-> Hello
- "Amazing!!! presentation Mr Akshay!!"-> Amazing

In [46]:
```
text1 = '"Hello there!" he said.'
text2 = 'Amazing!!! presentation Mr Akshay!!'
match1 = re.match(r'^[\s"\']*([A-Z][a-zA-Z]*)\b', text1)
if match1:
    first_word = match1.group(1)
    print(text1,", First word:", first_word)
match2 = re.match(r'^[\s"\']*([A-Z][a-zA-Z]*)\b', text2)
if match2:
    first_word = match2.group(1)
    print(text2,", First word:", first_word)
```

```
"Hello there!" he said. , First word: Hello
Amazing!!! presentation Mr Akshay!! , First word: Amazing
```

Comments