INFORMATION RETRIEVAL(CSE508)

ASSIGNMENT 1

Group Member :	Abhishek Nigam	**RollNo:** MT22003
	Dhananjay Bagul	**RollNo:** MT22028
	Wilson Radadia	**RollNo:** MT22091

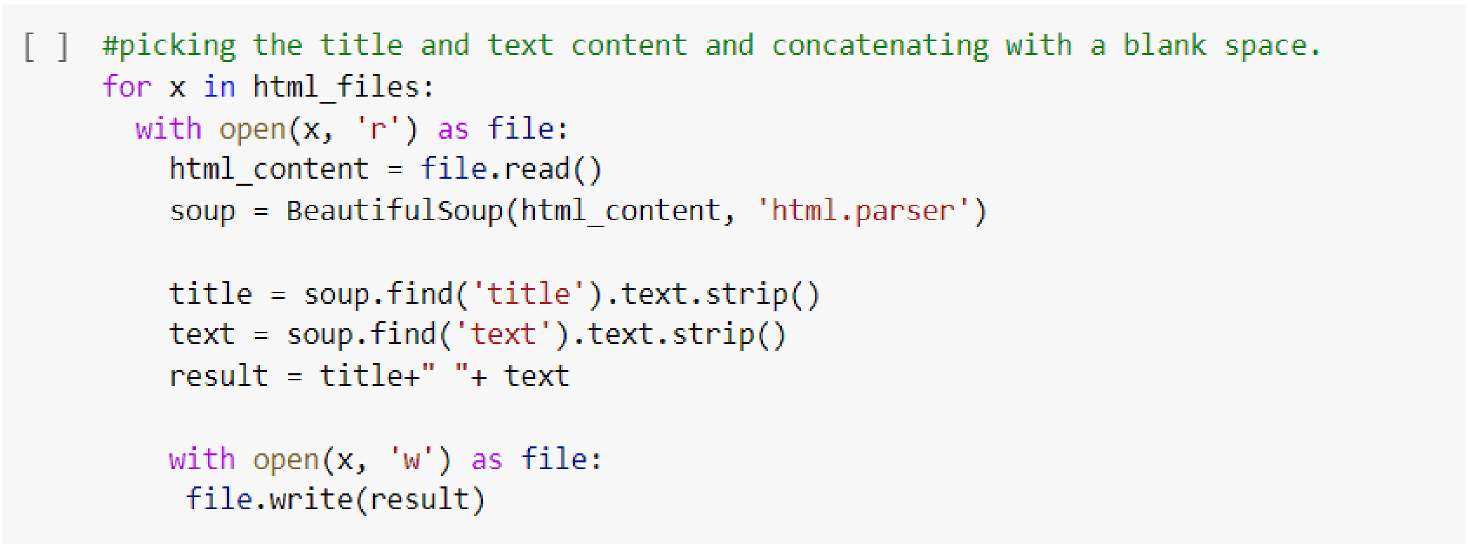
Question 1:

LIBRARIES USED:

1. **os**: for importing dataset folder from directory.
2. **nltk**: for performing operations like tokenization, stemming, detecting stop words etc.
3. **pandas**: for implementing dataFrames.
4. **string**: for working with text data present in dataset files.
5. **re**: Provides operations of detecting patterns with the help of regular expressions

(i) Relevant Text Extraction:

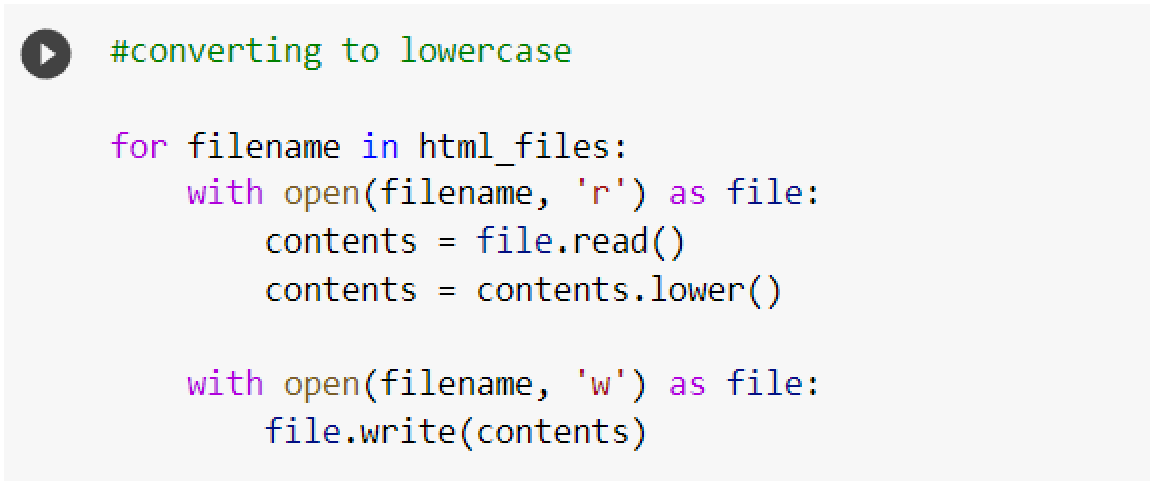
1. Used the bs4 from BeautifulSoup to concatenate the title and text.
2. Use the find() method to extract the title and text from the parsed HTML file.
3. Store the concatenated sentence into the result.
4. Finally make the changes in the original files.



```
[ ]  #picking the title and text content and concatenating with a blank space.
     for x in html_files:
       with open(x, 'r') as file:
         html_content = file.read()
         soup = BeautifulSoup(html_content, 'html.parser')

         title = soup.find('title').text.strip()
         text = soup.find('text').text.strip()
         result = title+" "+ text

         with open(x, 'w') as file:
          file.write(result)
```

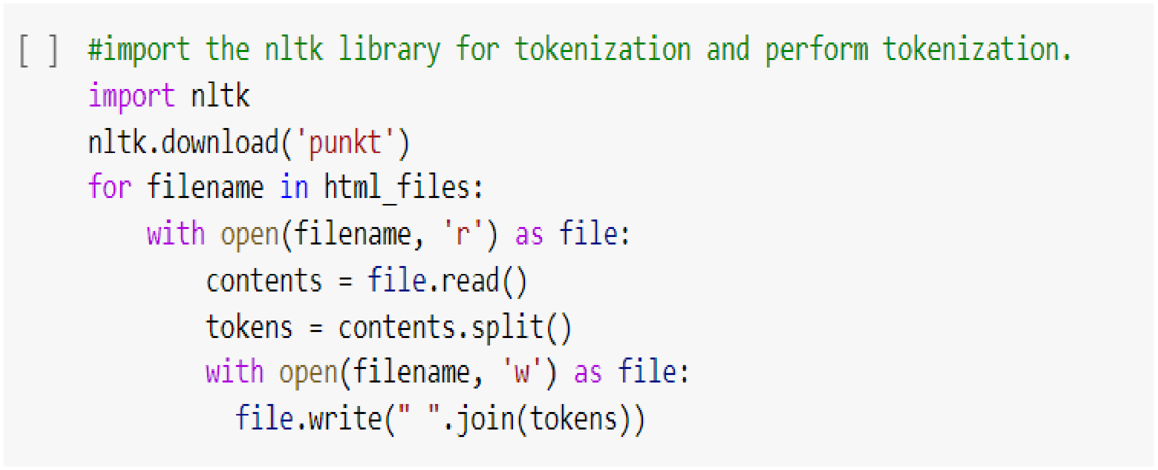
(ii) Preprocessing:

1.Converted the complete text in all the files to lower case using .lower() function.



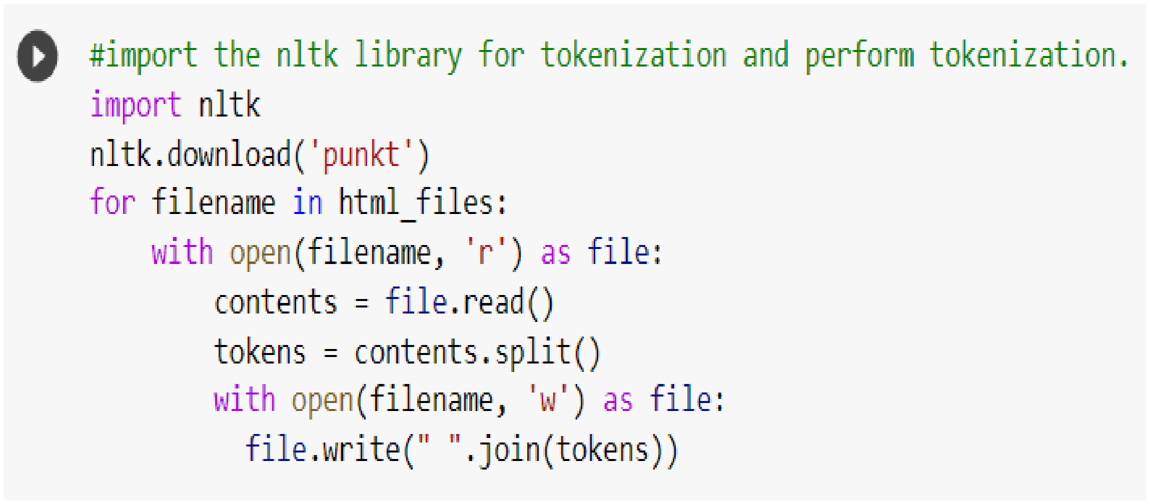
```python
#converting to lowercase

for filename in html_files:
    with open(filename, 'r') as file:
        contents = file.read()
        contents = contents.lower()

    with open(filename, 'w') as file:
        file.write(contents)
```

2. performed tokenization on all the files after converting to lowercase.



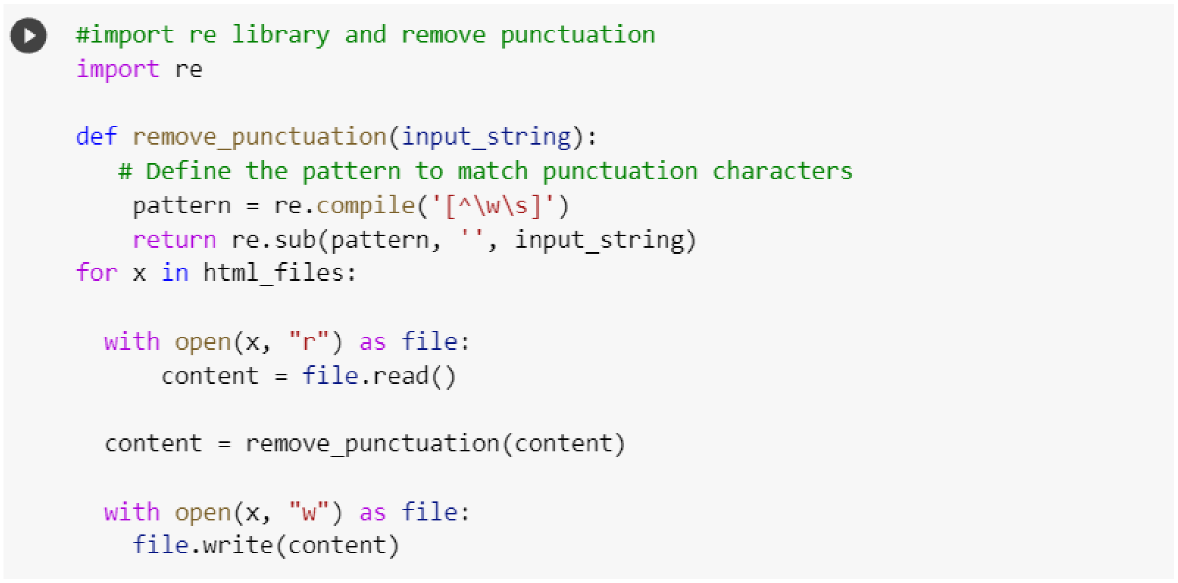
```python
[ ] #import the nltk library for tokenization and perform tokenization.
    import nltk
    nltk.download('punkt')
    for filename in html_files:
        with open(filename, 'r') as file:
            contents = file.read()
            tokens = contents.split()
            with open(filename, 'w') as file:
              file.write(" ".join(tokens))
```

3.Removed stopwords from the files after performing tokenization.



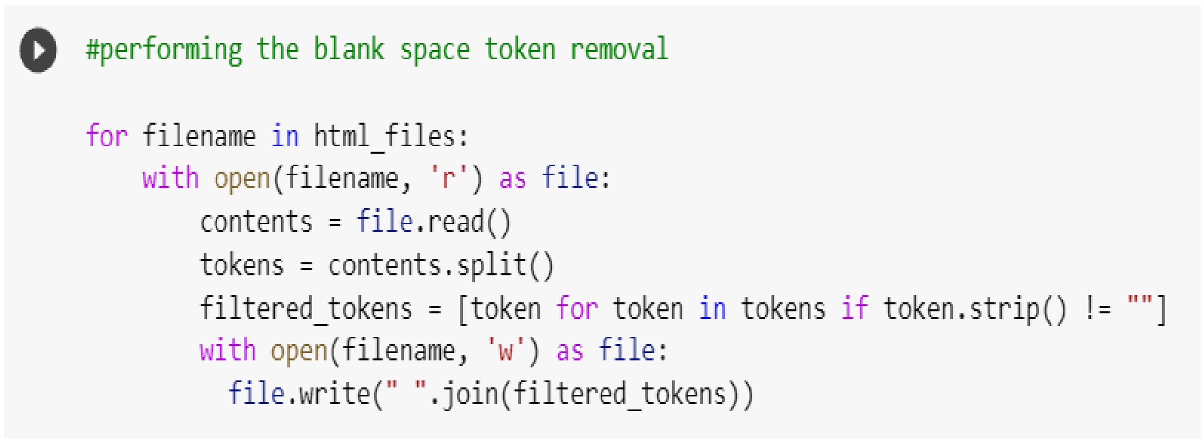
```python
#import the nltk library for tokenization and perform tokenization.
import nltk
nltk.download('punkt')
for filename in html_files:
    with open(filename, 'r') as file:
        contents = file.read()
        tokens = contents.split()
        with open(filename, 'w') as file:
          file.write(" ".join(tokens))
```

4. Used re.sub() to remove the punctuation marks from the text files.



```
#import re library and remove punctuation
import re

def remove_punctuation(input_string):
   # Define the pattern to match punctuation characters
    pattern = re.compile('[^\w\s]')
    return re.sub(pattern, '', input_string)
for x in html_files:

  with open(x, "r") as file:
      content = file.read()

  content = remove_punctuation(content)

  with open(x, "w") as file:
    file.write(content)
```

5. Then used .strip() to remove the blank spaces from all the files in the folder.

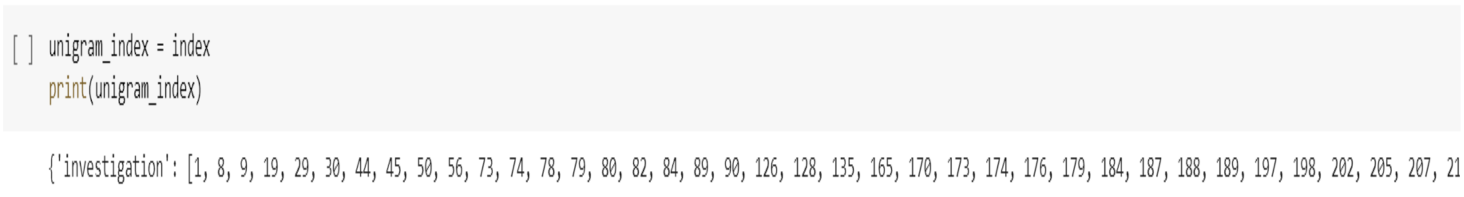


```
#performing the blank space token removal

for filename in html_files:
    with open(filename, 'r') as file:
        contents = file.read()
        tokens = contents.split()
        filtered_tokens = [token for token in tokens if token.strip() != ""]
        with open(filename, 'w') as file:
          file.write(" ".join(filtered_tokens))
```

Finally printed the first 5 files after performing all the operations after each step.

QUESTION 2

Unigram Index :



```
[ ] unigram_index = index
    print(unigram_index)

    {'investigation': [1, 8, 9, 19, 29, 30, 44, 45, 50, 56, 73, 74, 78, 79, 80, 82, 84, 89, 90, 126, 128, 135, 165, 170, 173, 174, 176, 179, 184, 187, 188, 189, 197, 198, 202, 205, 207, 21
```

FUNCTION DEFINITIONS FOR OR, AND, NOT, AND NOT and OR NOT

AND FUNCTION IMPLEMENTATION

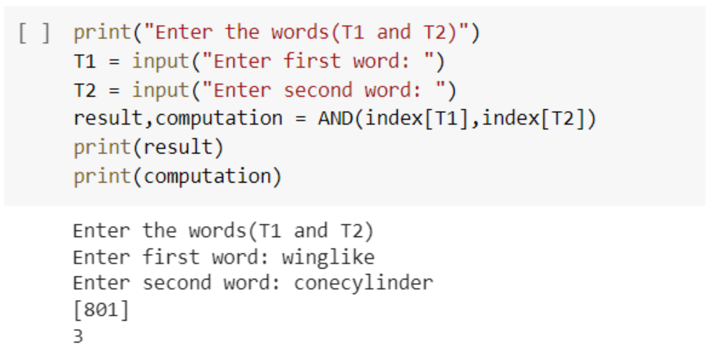
1. There is a list called output which stores the document names in which terms are matched. Initially the list is empty.

2. For the two posting lists termwise the documents are compared and the iterator is moved until one of the lists is exhausted.

3. If the document numbers are matched, the iterator moves forward putting the document number matched in the output list.

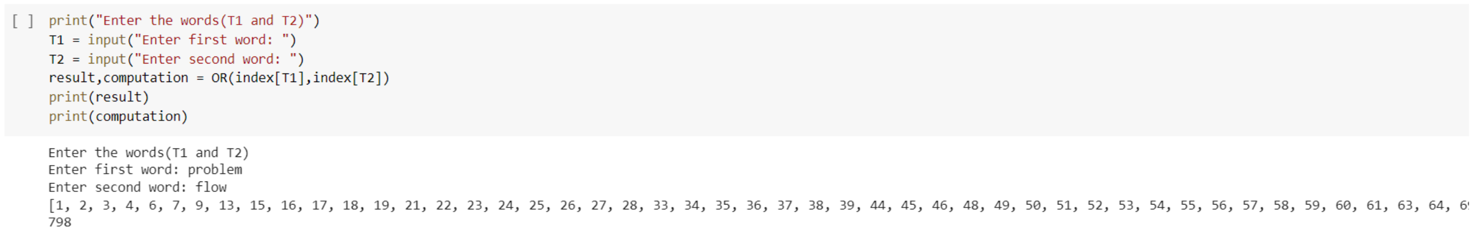
4. The number of comparisons are counted as the iterator moves and comparisons are done.

5. The output list storing the documents matched and the count of comparisons will be returned.



```
[ ] print("Enter the words(T1 and T2)")
    T1 = input("Enter first word: ")
    T2 = input("Enter second word: ")
    result,computation = AND(index[T1],index[T2])
    print(result)
    print(computation)

    Enter the words(T1 and T2)
    Enter first word: winglike
    Enter second word: conecylinder
    [801]
    3
```

OR FUNCTION IMPLEMENTATION



```
[ ] print("Enter the words(T1 and T2)")
    T1 = input("Enter first word: ")
    T2 = input("Enter second word: ")
    result,computation = OR(index[T1],index[T2])
    print(result)
    print(computation)

    Enter the words(T1 and T2)
    Enter first word: problem
    Enter second word: flow
    [1, 2, 3, 4, 6, 7, 9, 13, 15, 16, 17, 18, 19, 21, 22, 23, 24, 25, 26, 27, 28, 33, 34, 35, 36, 37, 38, 39, 44, 45, 46, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 63, 64, 6
    798
```

1. There is a list called output which stores the document names in which terms are matched. Initially the list is empty.

2. The document numbers are matched and if the document is matched the document is added to the output list and the iterator for both the posting lists is moved forward.

3. If the document numbers don't match the smaller document number is added to the output list and the iterator skips that document number in that particular posting list.

4. The number of comparisons are counted as the iterator moves and comparisons are done.

5. The output list storing the documents matched and the count of comparisons will be returned.

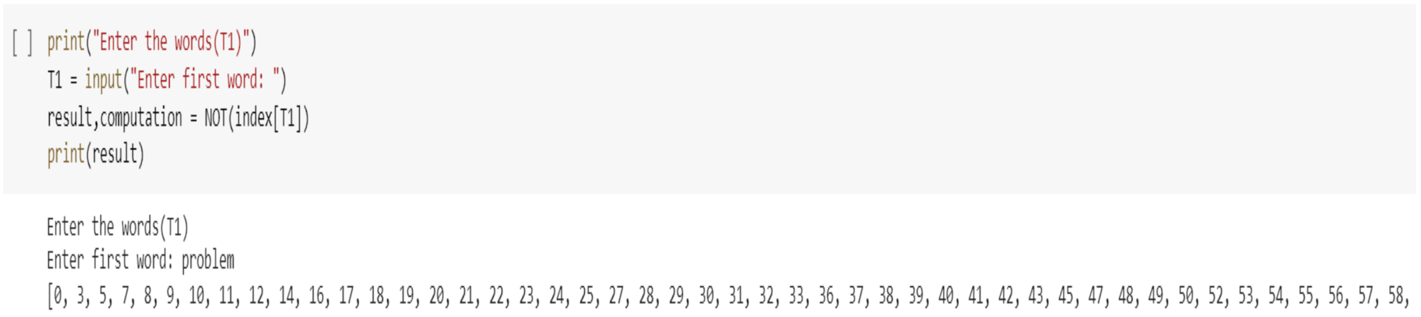
NOT FUNCTION IMPLEMENTATION

1. The NOT function returns those documents which do not contain the terms or those not present in the posting list.

2. The document numbers present are matched with the document numbers present in the posting lists and the count for comparisons is increased.

3. Those document count which are not present in the posting list are added to the output list.

4. If all the document numbers are present in the posting lists, then output list will not contain any document so it will return 0



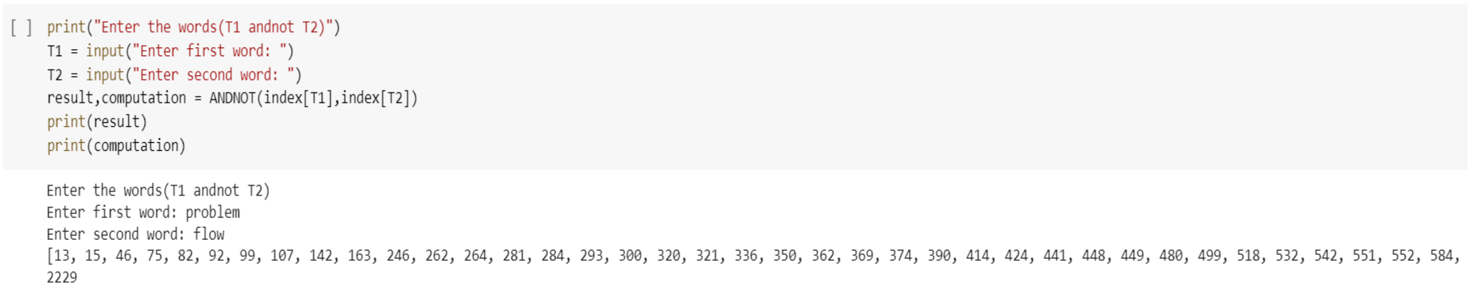
```
[ ] print("Enter the words(T1)")
    T1 = input("Enter first word: ")
    result,computation = NOT(index[T1])
    print(result)

    Enter the words(T1)
    Enter first word: problem
    [0, 3, 5, 7, 8, 9, 10, 11, 12, 14, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 27, 28, 29, 30, 31, 32, 33, 36, 37, 38, 39, 40, 41, 42, 43, 45, 47, 48, 49, 50, 52, 53, 54, 55, 56, 57, 58,
```

ANDNOT FUNCTION IMPLEMENTATION

1. AND NOT function is the combination of AND and NOT performed together.

2. At first the NOT function is implemented on one of the posting lists using the NOT function.

3. After that the AND function implementation is applied on the posting list1 and **not posting list** obtained from not function and the output list contains the result of the combination. The comparisons are the total count of NOT and AND function implementation.



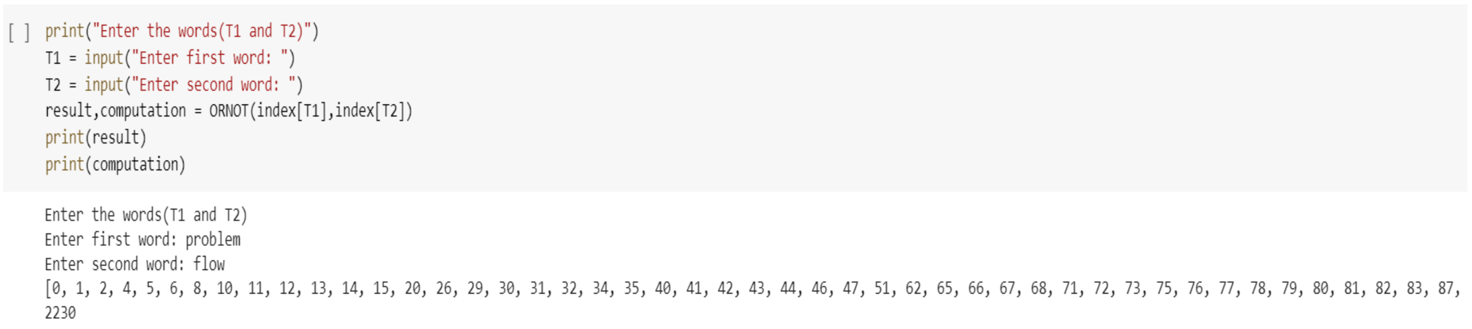
```
[ ] print("Enter the words(T1 andnot T2)")
    T1 = input("Enter first word: ")
    T2 = input("Enter second word: ")
    result,computation = ANDNOT(index[T1],index[T2])
    print(result)
    print(computation)

    Enter the words(T1 andnot T2)
    Enter first word: problem
    Enter second word: flow
    [13, 15, 46, 75, 82, 92, 99, 107, 142, 163, 246, 262, 264, 281, 284, 293, 300, 320, 321, 336, 350, 362, 369, 374, 390, 414, 424, 441, 448, 449, 480, 499, 518, 532, 542, 551, 552, 584,
    2229
```

ORNOT FUNCTION IMPLEMENTATION

1. OR NOT function is the combination of OR and NOT performed together.

2. At first the NOT function is implemented on one of the posting lists using the NOT function.

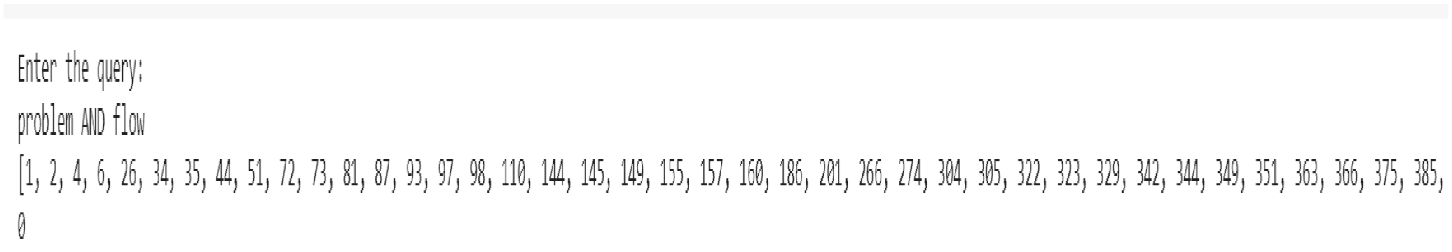
3. After that the OR function implementation is applied on the posting list1 and **not posting list** obtained from not function and the output list contains the result of the combination. The comparisons are the total count of NOT and OR function implementation.



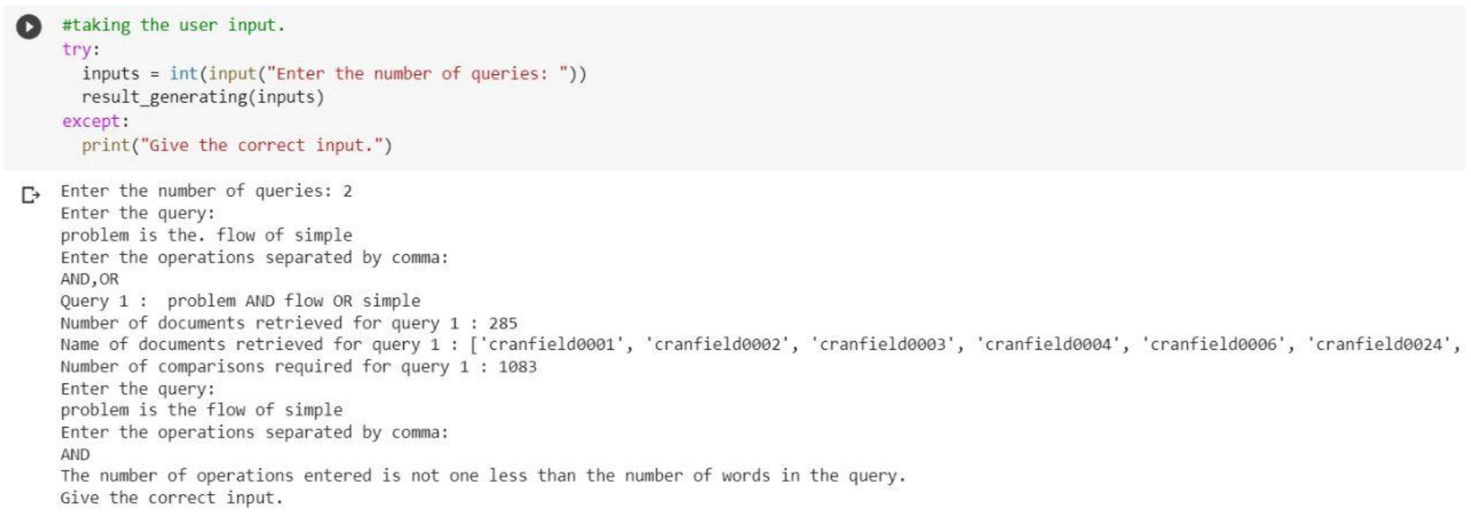
```
[ ] print("Enter the words(T1 and T2)")
    T1 = input("Enter first word: ")
    T2 = input("Enter second word: ")
    result,computation = ORNOT(index[T1],index[T2])
    print(result)
    print(computation)
```

```
Enter the words(T1 and T2)
Enter first word: problem
Enter second word: flow
[0, 1, 2, 4, 5, 6, 8, 10, 11, 12, 13, 14, 15, 20, 26, 29, 30, 31, 32, 34, 35, 40, 41, 42, 43, 44, 46, 47, 51, 62, 65, 66, 67, 68, 71, 72, 73, 75, 76, 77, 78, 79, 80, 81, 82, 83, 87,
2230
```

Generalized Query



```
Enter the query:
problem AND flow
[1, 2, 4, 6, 26, 34, 35, 44, 51, 72, 73, 81, 87, 93, 97, 98, 110, 144, 145, 149, 155, 157, 160, 186, 201, 266, 274, 304, 305, 322, 323, 329, 342, 344, 349, 351, 363, 366, 375, 385,
0
```



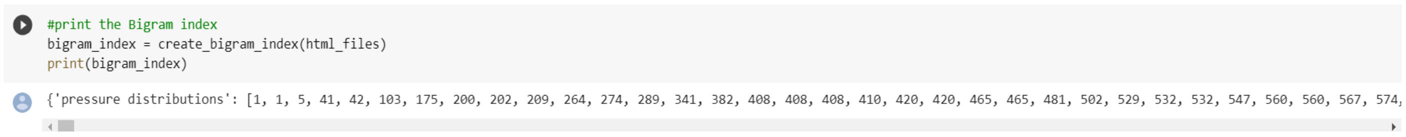
```
#taking the user input.
try:
  inputs = int(input("Enter the number of queries: "))
  result_generating(inputs)
except:
  print("Give the correct input.")
```

```
Enter the number of queries: 2
Enter the query:
problem is the. flow of simple
Enter the operations separated by comma:
AND,OR
Query 1 :  problem AND flow OR simple
Number of documents retrieved for query 1 : 285
Name of documents retrieved for query 1 : ['cranfield0001', 'cranfield0002', 'cranfield0003', 'cranfield0004', 'cranfield0006', 'cranfield0024',
Number of comparisons required for query 1 : 1083
Enter the query:
problem is the flow of simple
Enter the operations separated by comma:
AND
The number of operations entered is not one less than the number of words in the query.
Give the correct input.
```

QUESTION 3

i) Bigram Inverted Index:

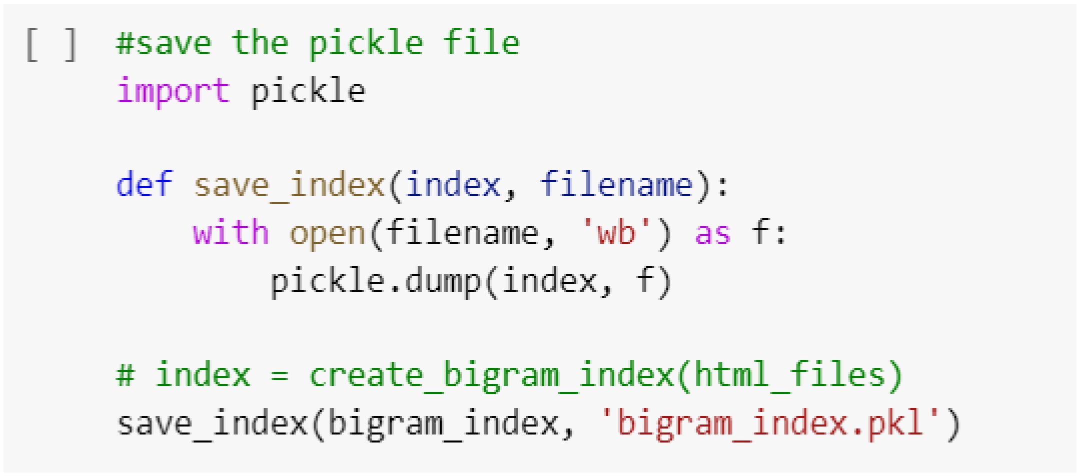
1.



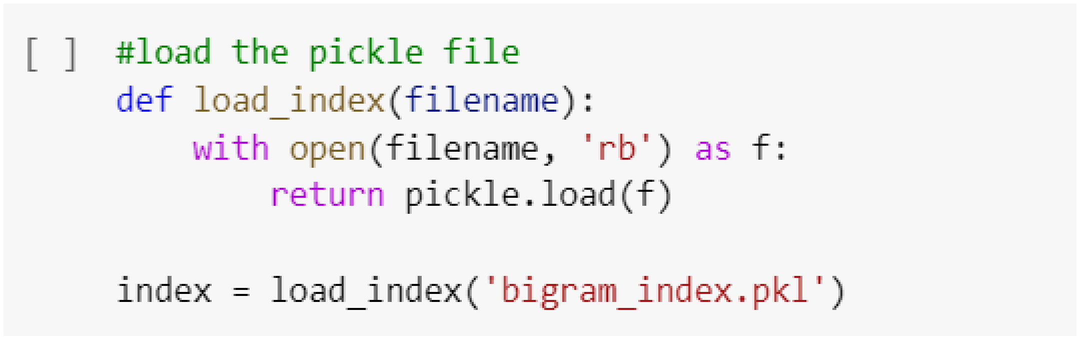
```
#print the Bigram index
bigram_index = create_bigram_index(html_files)
print(bigram_index)
```

```
{'pressure distributions': [1, 1, 5, 41, 42, 103, 175, 200, 202, 209, 264, 274, 289, 341, 382, 408, 408, 408, 410, 420, 420, 465, 465, 481, 502, 529, 532, 532, 547, 560, 560, 567, 574,
```

2.

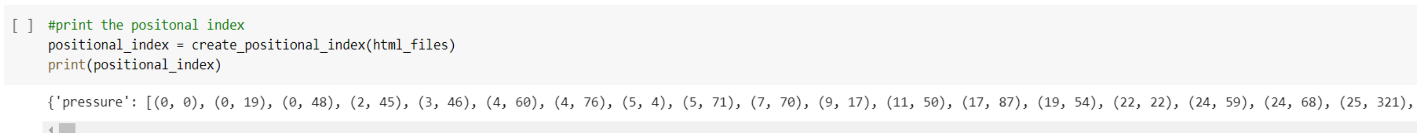


```
[ ]  #save the pickle file
     import pickle

     def save_index(index, filename):
         with open(filename, 'wb') as f:
             pickle.dump(index, f)

     # index = create_bigram_index(html_files)
     save_index(bigram_index, 'bigram_index.pkl')
```

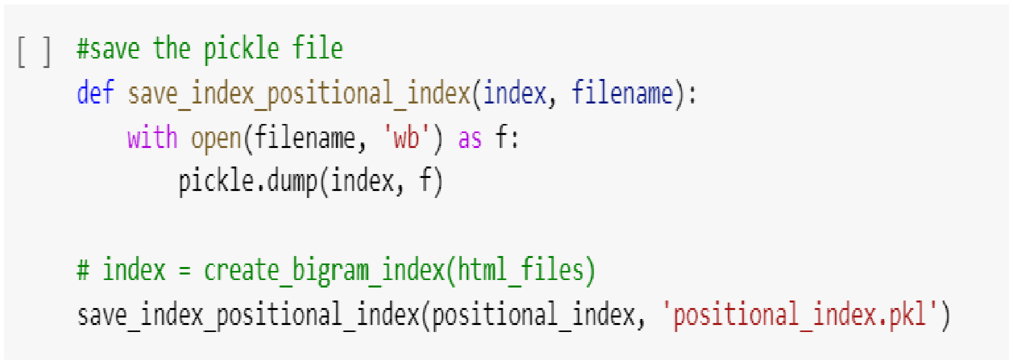


```
[ ]  #load the pickle file
     def load_index(filename):
         with open(filename, 'rb') as f:
             return pickle.load(f)

     index = load_index('bigram_index.pkl')
```

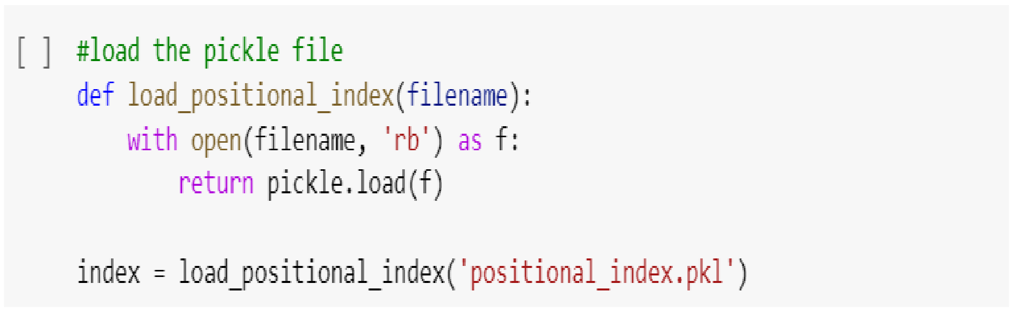
ii) Positional index:

1. 

```
[ ] #print the positonal index
    positional_index = create_positional_index(html_files)
    print(positional_index)

    {'pressure': [(0, 0), (0, 19), (0, 48), (2, 45), (3, 46), (4, 60), (4, 76), (5, 4), (5, 71), (7, 70), (9, 17), (11, 50), (17, 87), (19, 54), (22, 22), (24, 59), (24, 68), (25, 321),
```

2. 

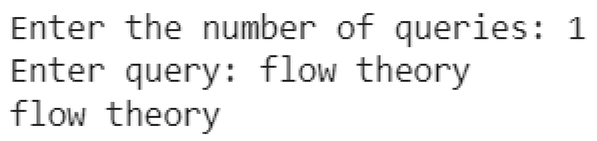
```
[ ] #save the pickle file
    def save_index_positional_index(index, filename):
        with open(filename, 'wb') as f:
            pickle.dump(index, f)

    # index = create_bigram_index(html_files)
    save_index_positional_index(positional_index, 'positional_index.pkl')
```



```
[ ] #load the pickle file
    def load_positional_index(filename):
        with open(filename, 'rb') as f:
            return pickle.load(f)

    index = load_positional_index('positional_index.pkl')
```

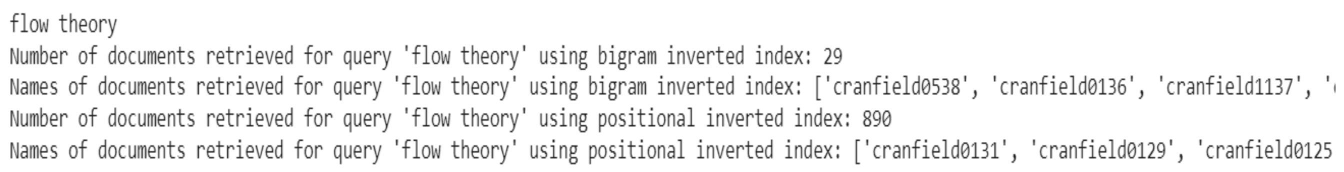
iii) Compare and comment on your results using (i) and (ii).

1. Input Format:



```
Enter the number of queries: 1
Enter query: flow theory
flow theory
```

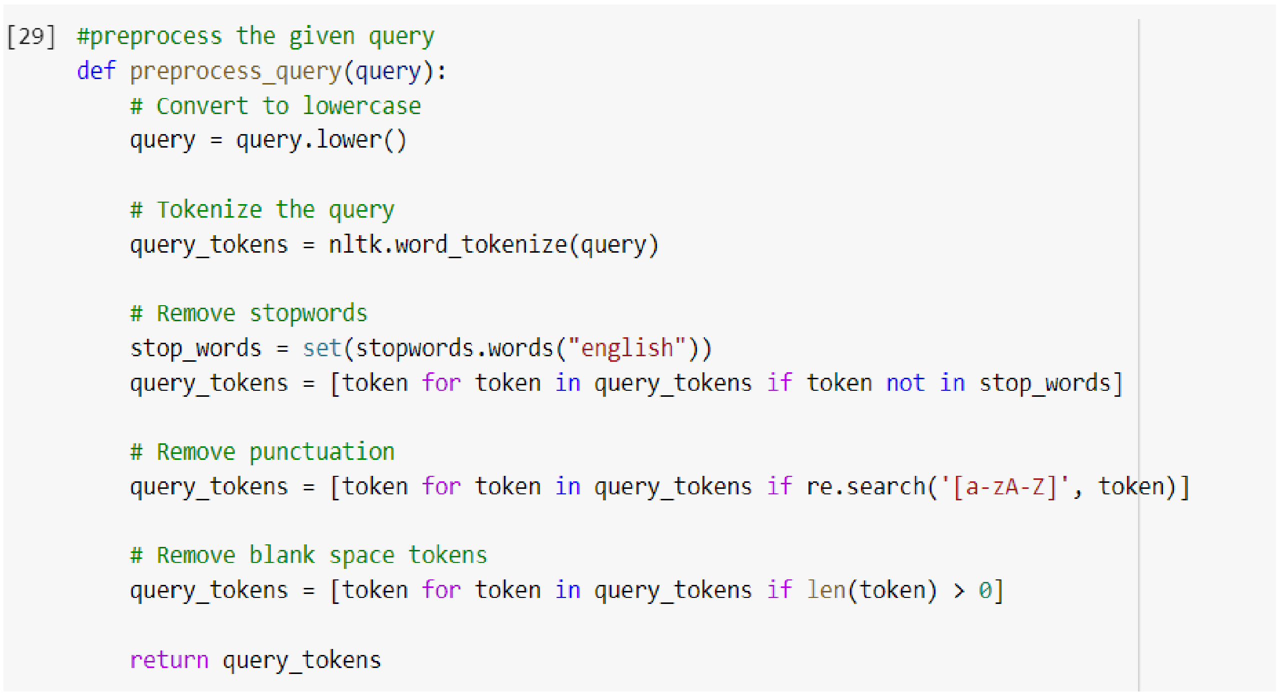
2. Output Format:



```
flow theory
Number of documents retrieved for query 'flow theory' using bigram inverted index: 29
Names of documents retrieved for query 'flow theory' using bigram inverted index: ['cranfield0538', 'cranfield0136', 'cranfield1137', '
Number of documents retrieved for query 'flow theory' using positional inverted index: 890
Names of documents retrieved for query 'flow theory' using positional inverted index: ['cranfield0131', 'cranfield0129', 'cranfield0125
```

In this the documents retrieved for query **flow theory** using bigram inverted index and positional inverted index output is given as we can see the numbers are different for both as in bigram inverted index it checks for two words at once which is not the case for positional inverted index.

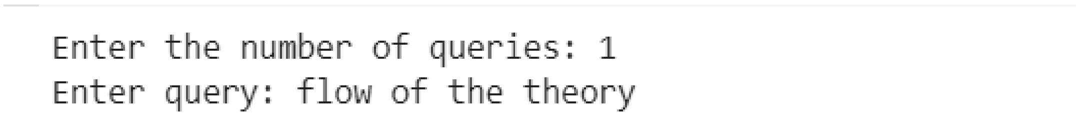
3. Preprocessing:



```
[29] #preprocess the given query
     def preprocess_query(query):
         # Convert to lowercase
         query = query.lower()

         # Tokenize the query
         query_tokens = nltk.word_tokenize(query)

         # Remove stopwords
         stop_words = set(stopwords.words("english"))
         query_tokens = [token for token in query_tokens if token not in stop_words]

         # Remove punctuation
         query_tokens = [token for token in query_tokens if re.search('[a-zA-Z]', token)]

         # Remove blank space tokens
         query_tokens = [token for token in query_tokens if len(token) > 0]

         return query_tokens
```

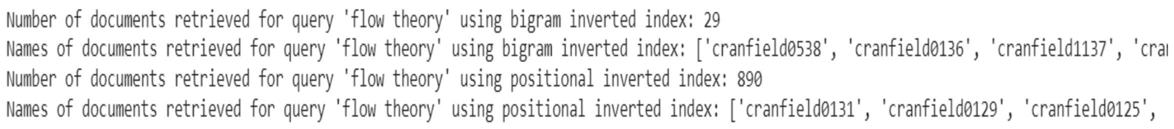
4. Sample Test Case:

a. Input



```
Enter the number of queries: 1
Enter query: flow of the theory
```

b.Output:



```
Number of documents retrieved for query 'flow theory' using bigram inverted index: 29
Names of documents retrieved for query 'flow theory' using bigram inverted index: ['cranfield0538', 'cranfield0136', 'cranfield1137', 'cra
Number of documents retrieved for query 'flow theory' using positional inverted index: 890
Names of documents retrieved for query 'flow theory' using positional inverted index: ['cranfield0131', 'cranfield0129', 'cranfield0125',
```