NAME: VANSHAJ SHARMA

ROLLNO: MT23103

COURSE: M.TECH CSE

YEAR: FIRST YEAR

Q1

Approach :

In this Question, I used libraries such as nltk, pathlib, bs4 and os. Nltk is used to provide functions for natural language processing like word tokenizer, stemming and lemmatization etc. For this question I performed following steps in the following order.

a. Lowercase the text

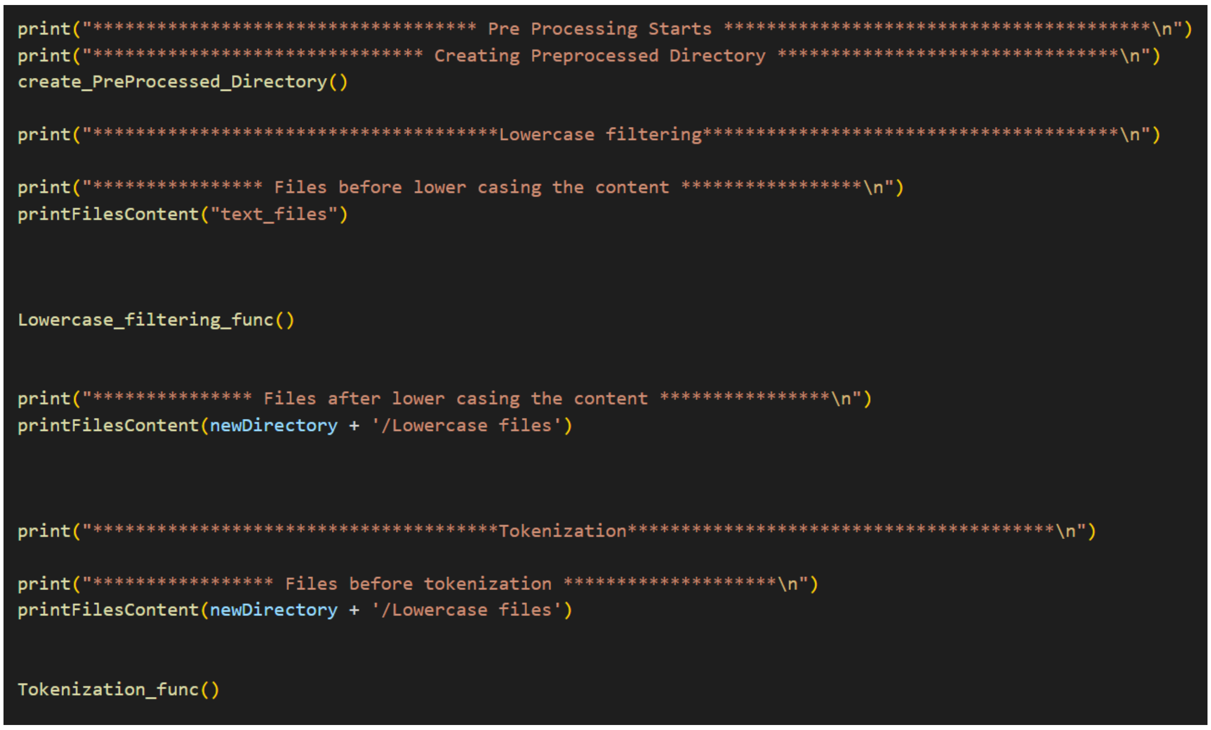
b. Perform tokenization

c. Remove stopwords

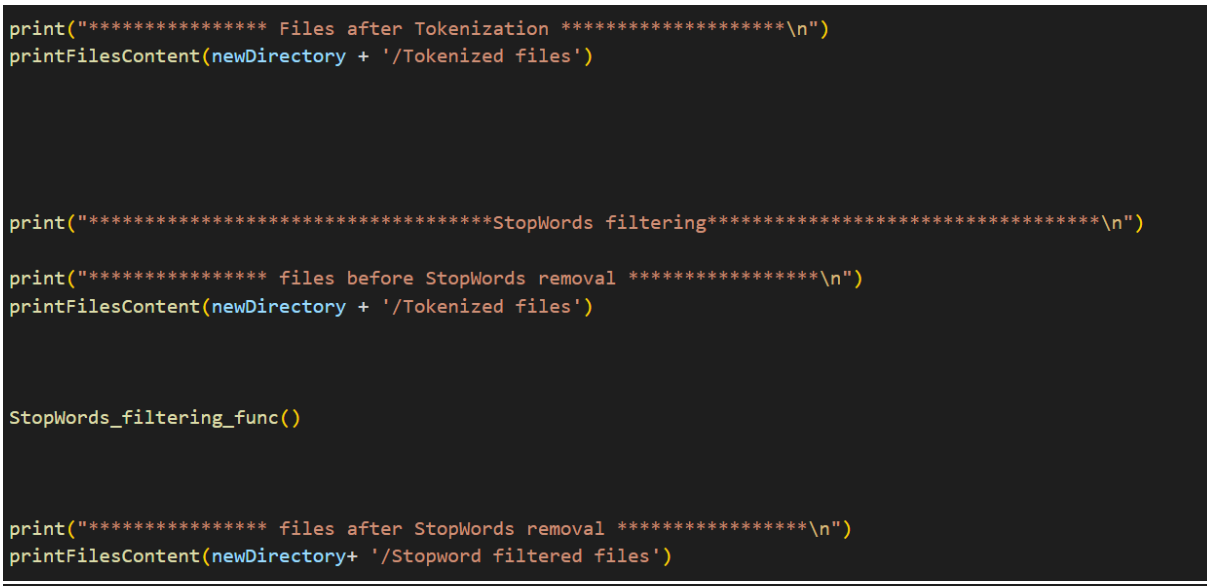
d. Remove punctuations

e. Remove blank space tokens

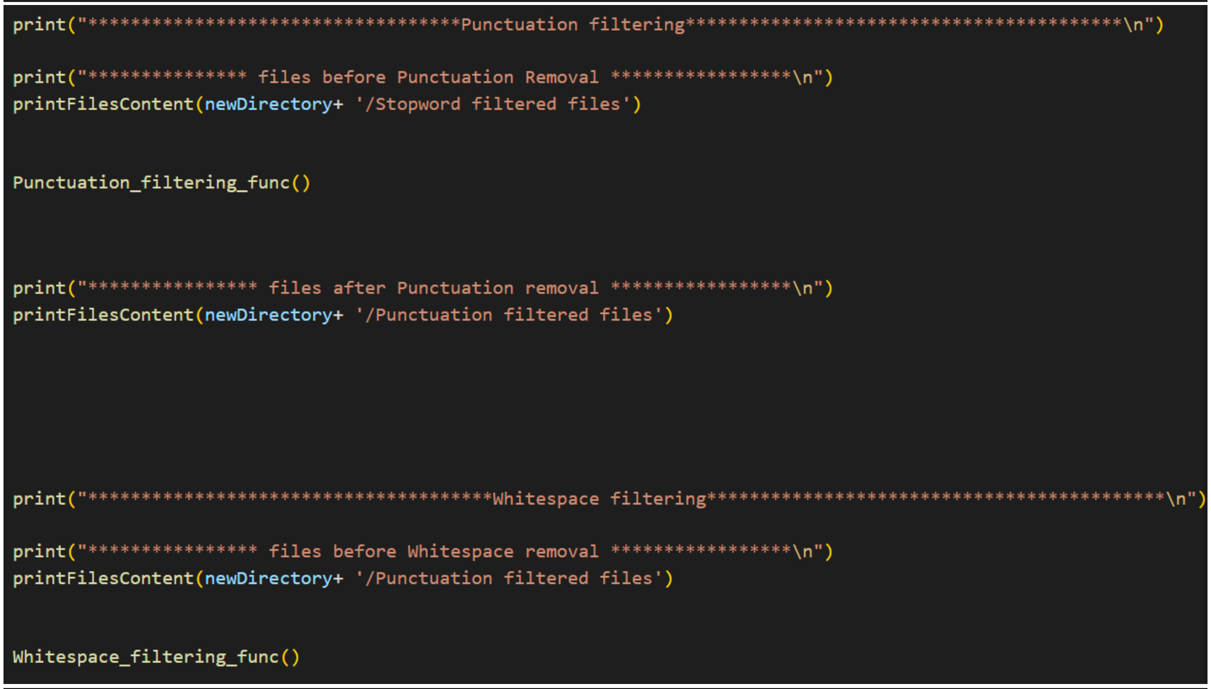
After each step, I am saving resultant files in the corresponding folder to these functions. Here each step is taking input the output of previous step and then processing giving the results.
Before and after each step, we are printing all the files so that we could be able to see the changes have been done by each function. Order of processing is :-



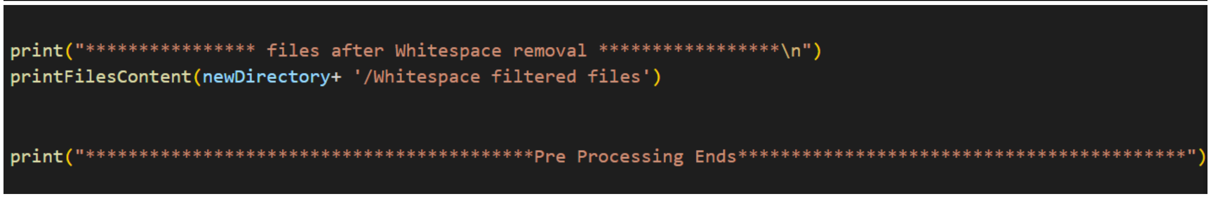
```
print("************************************ Pre Processing Starts ****************************************\n")
print("******************************* Creating Preprocessed Directory ********************************\n")
create_PreProcessed_Directory()

print("**************************************Lowercase filtering***************************************\n")

print("**************** Files before lower casing the content *****************\n")
printFilesContent("text_files")



Lowercase_filtering_func()


print("*************** Files after lower casing the content ****************\n")
printFilesContent(newDirectory + '/Lowercase files')



print("**************************************Tokenization****************************************\n")

print("***************** Files before tokenization ********************\n")
printFilesContent(newDirectory + '/Lowercase files')


Tokenization_func()
```



```
print("**************** Files after Tokenization ********************\n")
printFilesContent(newDirectory + '/Tokenized files')




print("************************************StopWords filtering***********************************\n")

print("**************** files before StopWords removal *****************\n")
printFilesContent(newDirectory + '/Tokenized files')



StopWords_filtering_func()



print("**************** files after StopWords removal *****************\n")
printFilesContent(newDirectory+ '/Stopword filtered files')
```



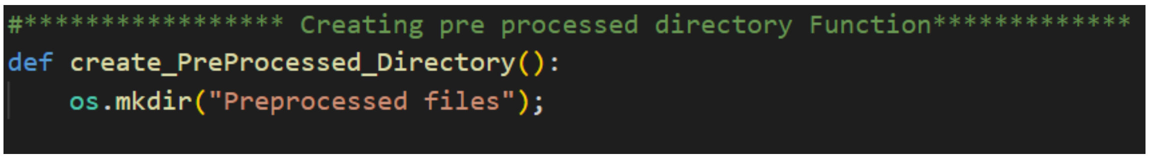
```
print("*********************************Punctuation filtering****************************************\n")

print("*************** files before Punctuation Removal ****************\n")
printFilesContent(newDirectory+ '/Stopword filtered files')


Punctuation_filtering_func()



print("**************** files after Punctuation removal ****************\n")
printFilesContent(newDirectory+ '/Punctuation filtered files')





print("************************************Whitespace filtering*****************************************\n")

print("**************** files before Whitespace removal ****************\n")
printFilesContent(newDirectory+ '/Punctuation filtered files')


Whitespace_filtering_func()
```



```
print("**************** files after Whitespace removal ****************\n")
printFilesContent(newDirectory+ '/Whitespace filtered files')


print("*****************************************Pre Processing Ends******************************************")
```

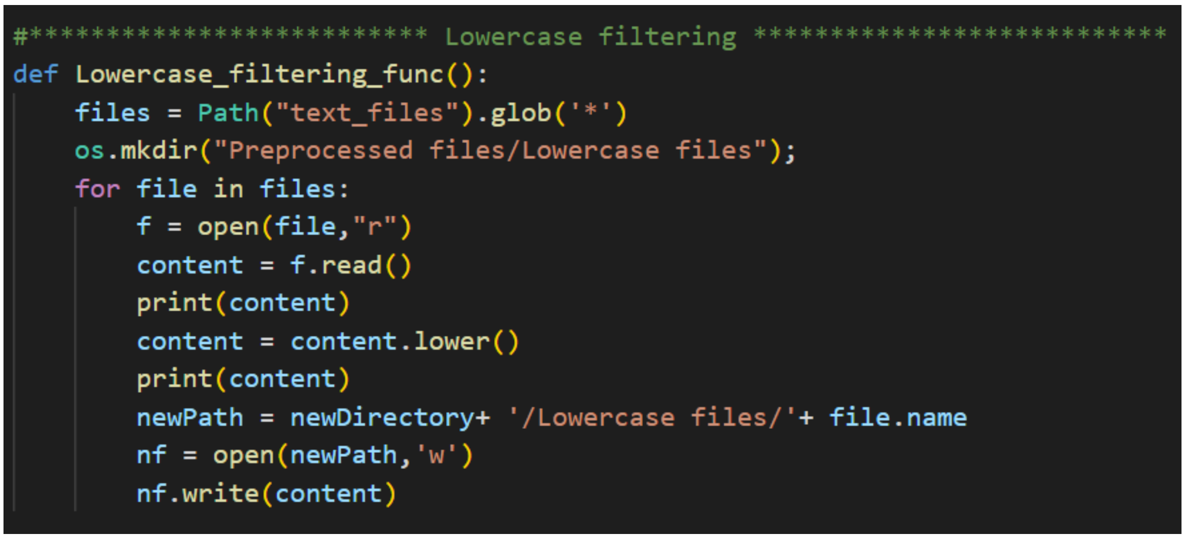
Methodologies :

First , I am creating a directory called pre processed files. In which the resultant folder for each pre processing step will be there.



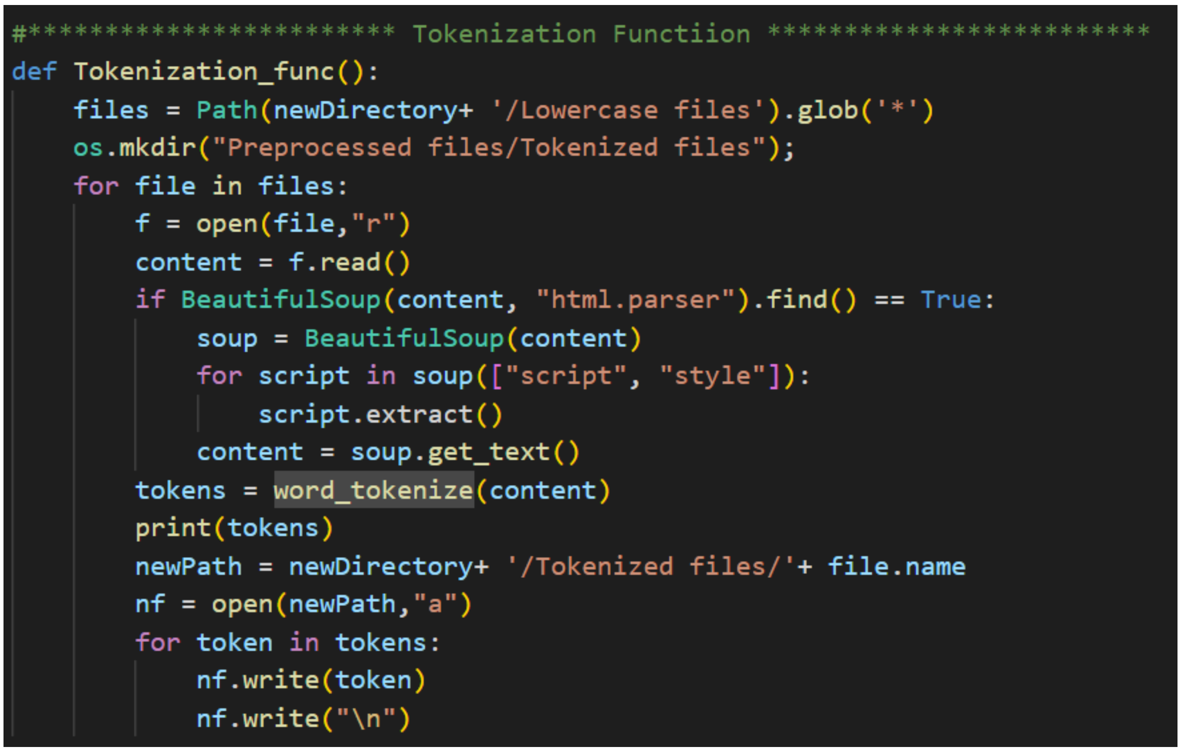
```
#***************** Creating pre processed directory Function*************
def create_PreProcessed_Directory():
    os.mkdir("Preprocessed files");
```

a. Lowercase the text: I created a function of this step, in which I first created the directory called lowercase files inside the pre-processed files directory then I am reading all the 999 files given in the assignment and converting their content in lowercase and saving the content in the new 999 files inside the lowercase directory.



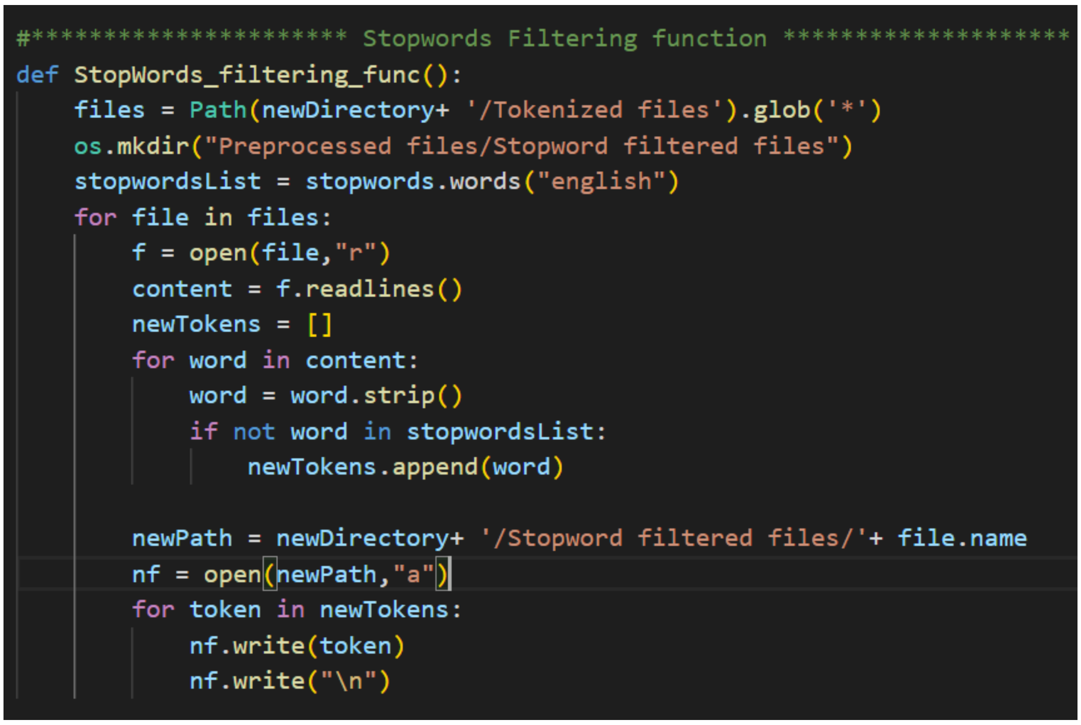
```python
#************************* Lowercase filtering **************************
def Lowercase_filtering_func():
    files = Path("text_files").glob('*')
    os.mkdir("Preprocessed files/Lowercase files");
    for file in files:
        f = open(file,"r")
        content = f.read()
        print(content)
        content = content.lower()
        print(content)
        newPath = newDirectory+ '/Lowercase files/'+ file.name
        nf = open(newPath,'w')
        nf.write(content)
```

b. Perform tokenization: In this function, I created a directory called Tokenized files. In which I am storing the 999 files which are having tokens of the text in 999 files that we produced for previous step i.e lowercase filtering. I used function word_tokenize from nltk library for tokenization.



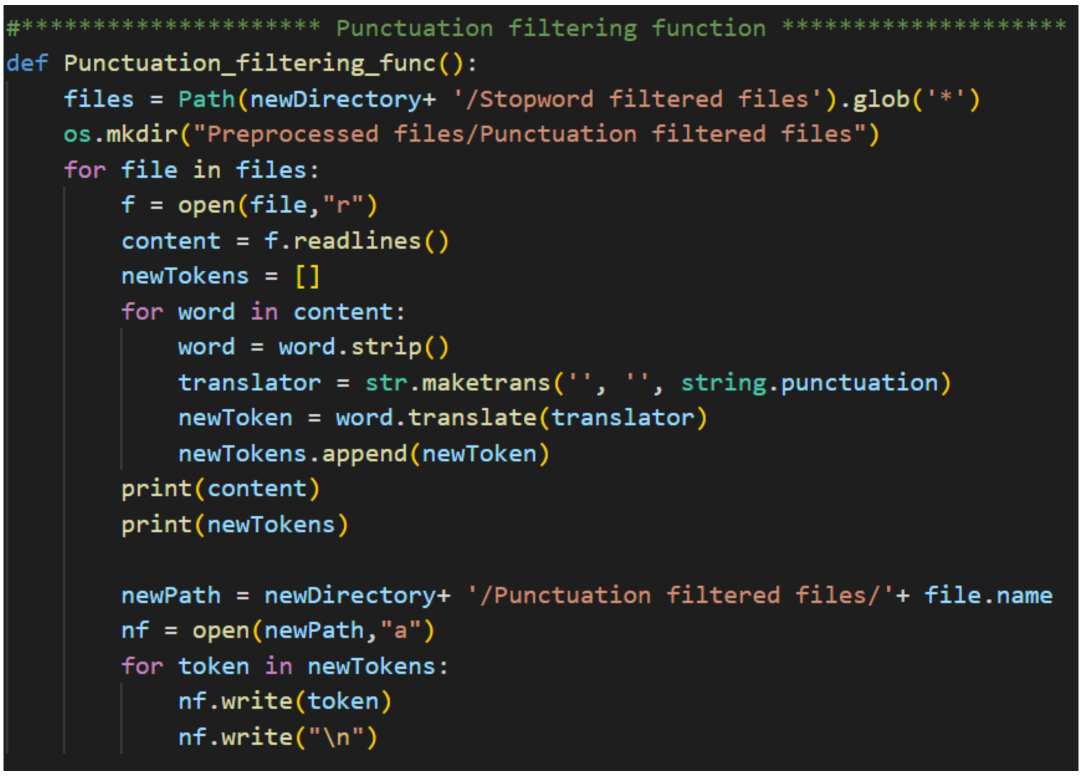
```python
#************************ Tokenization Functiion *************************
def Tokenization_func():
    files = Path(newDirectory+ '/Lowercase files').glob('*')
    os.mkdir("Preprocessed files/Tokenized files");
    for file in files:
        f = open(file,"r")
        content = f.read()
        if BeautifulSoup(content, "html.parser").find() == True:
            soup = BeautifulSoup(content)
            for script in soup(["script", "style"]):
                script.extract()
            content = soup.get_text()
        tokens = word_tokenize(content)
        print(tokens)
        newPath = newDirectory+ '/Tokenized files/'+ file.name
        nf = open(newPath,"a")
        for token in tokens:
            nf.write(token)
            nf.write("\n")
```

c. Remove stopwords: In the tokens that we generated for each file of 999 files. There are some tokens that are irrelevant that are not providing much meaning to the text. I created a directory for this step called stop word filtering files . So, using stop words list provided by the nltk library, I will parse each file among 999 files from tokenization directory and create corresponding new file in the stop word filtered files directory by saving all the tokens that are not in stop word list.



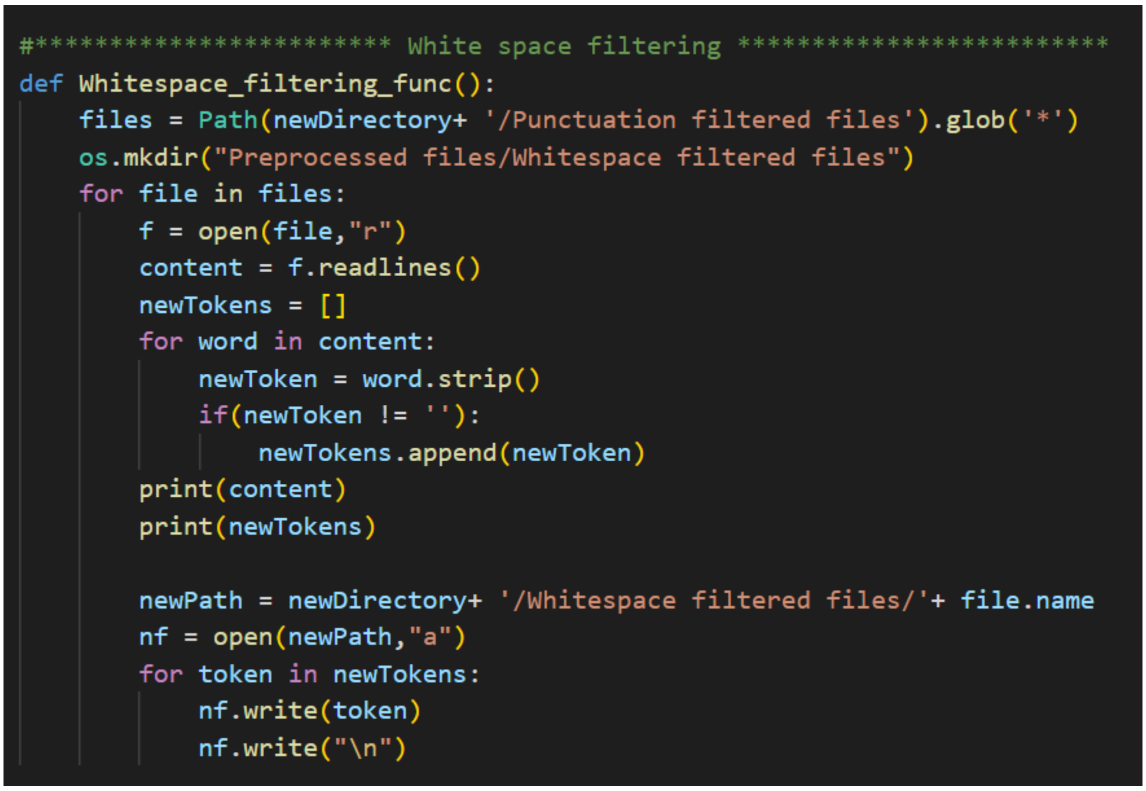
```python
#********************* Stopwords Filtering function *******************
def StopWords_filtering_func():
    files = Path(newDirectory+ '/Tokenized files').glob('*')
    os.mkdir("Preprocessed files/Stopword filtered files")
    stopwordsList = stopwords.words("english")
    for file in files:
        f = open(file,"r")
        content = f.readlines()
        newTokens = []
        for word in content:
            word = word.strip()
            if not word in stopwordsList:
                newTokens.append(word)

        newPath = newDirectory+ '/Stopword filtered files/'+ file.name
        nf = open(newPath,"a")
        for token in newTokens:
            nf.write(token)
            nf.write("\n")
```

d. Remove punctuations: For this step, I created a directory called punctuation filtered files. In which I stored the new files with the content of files in stop word filtering files after removing punctuation from these files. For removing punctuation, I used String class's translator and maketrans function with parameter "","" in it.



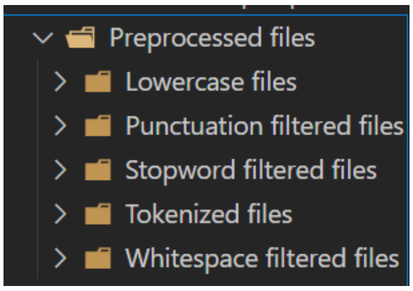
```python
#******************** Punctuation filtering function *******************
def Punctuation_filtering_func():
    files = Path(newDirectory+ '/Stopword filtered files').glob('*')
    os.mkdir("Preprocessed files/Punctuation filtered files")
    for file in files:
        f = open(file,"r")
        content = f.readlines()
        newTokens = []
        for word in content:
            word = word.strip()
            translator = str.maketrans('', '', string.punctuation)
            newToken = word.translate(translator)
            newTokens.append(newToken)
        print(content)
        print(newTokens)

        newPath = newDirectory+ '/Punctuation filtered files/'+ file.name
        nf = open(newPath,"a")
        for token in newTokens:
            nf.write(token)
            nf.write("\n")
```

e. Remove blank space tokens: For performing this step, I created a new folder for this step called Whitespace filtered files. I used resultant files from previous step and removed white space from the tokens list in these files and created new files from the tokens as content without white space in white space directory. For this I just checked whether the this token is whitespace or not.



```python
#************************ White space filtering ************************
def Whitespace_filtering_func():
    files = Path(newDirectory+ '/Punctuation filtered files').glob('*')
    os.mkdir("Preprocessed files/Whitespace filtered files")
    for file in files:
        f = open(file,"r")
        content = f.readlines()
        newTokens = []
        for word in content:
            newToken = word.strip()
            if(newToken != ''):
                newTokens.append(newToken)
        print(content)
        print(newTokens)

        newPath = newDirectory+ '/Whitespace filtered files/'+ file.name
        nf = open(newPath,"a")
        for token in newTokens:
            nf.write(token)
            nf.write("\n")
```

Results:



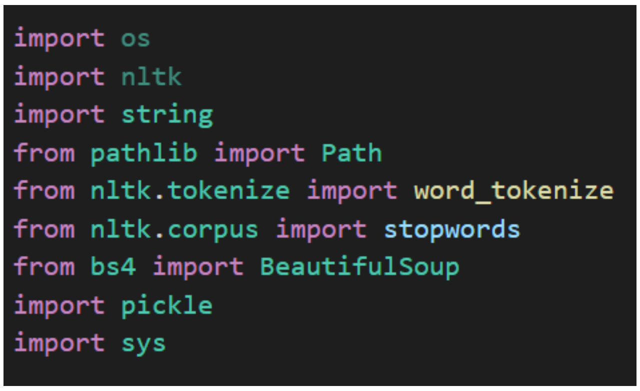
```
v Preprocessed files
  > Lowercase files
  > Punctuation filtered files
  > Stopword filtered files
  > Tokenized files
  > Whitespace filtered files
```

CLI output is very large so can’t mention it here, please run the Q1 file and see the output.

Q2

Approach:

Libraries used

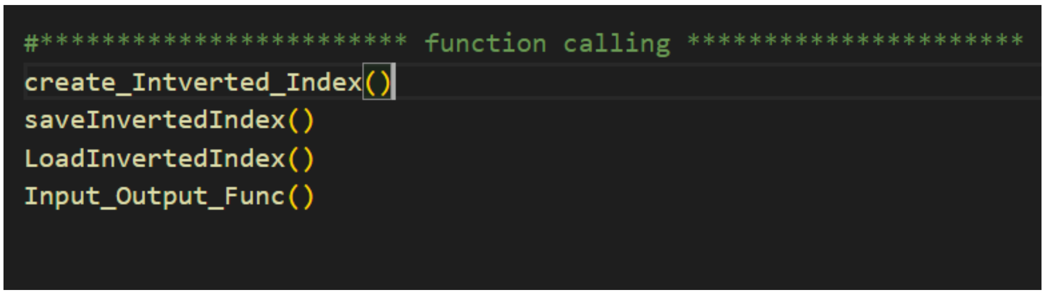


```python
import os
import nltk
import string
from pathlib import Path
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from bs4 import BeautifulSoup
import pickle
import sys
```

This question is about to create inverted index , So I created inverted index from the tokens that are generated for each file in question 1. I considered the token in files as terms and the file no is document id. Here I am making dictionary of key and value pair . In which key is the term and value is list of document id in which key appears. for performing operations on Query like AND, OR, AND NOT and OR NOT. I created functions that receive lists in the argument and then this function performs intersection or function related thing to find answer and return the list of documents id for the query. Q2 program take input from user the Query and the operations in needed to be performed on the query then it will preprocess the Query with the same method as we did in Question 1. We will process the query in the left to right in terms of operators. Used pickle library to save the inverted index into byte format and load the byte format inverted index into dictionary format inverted index.

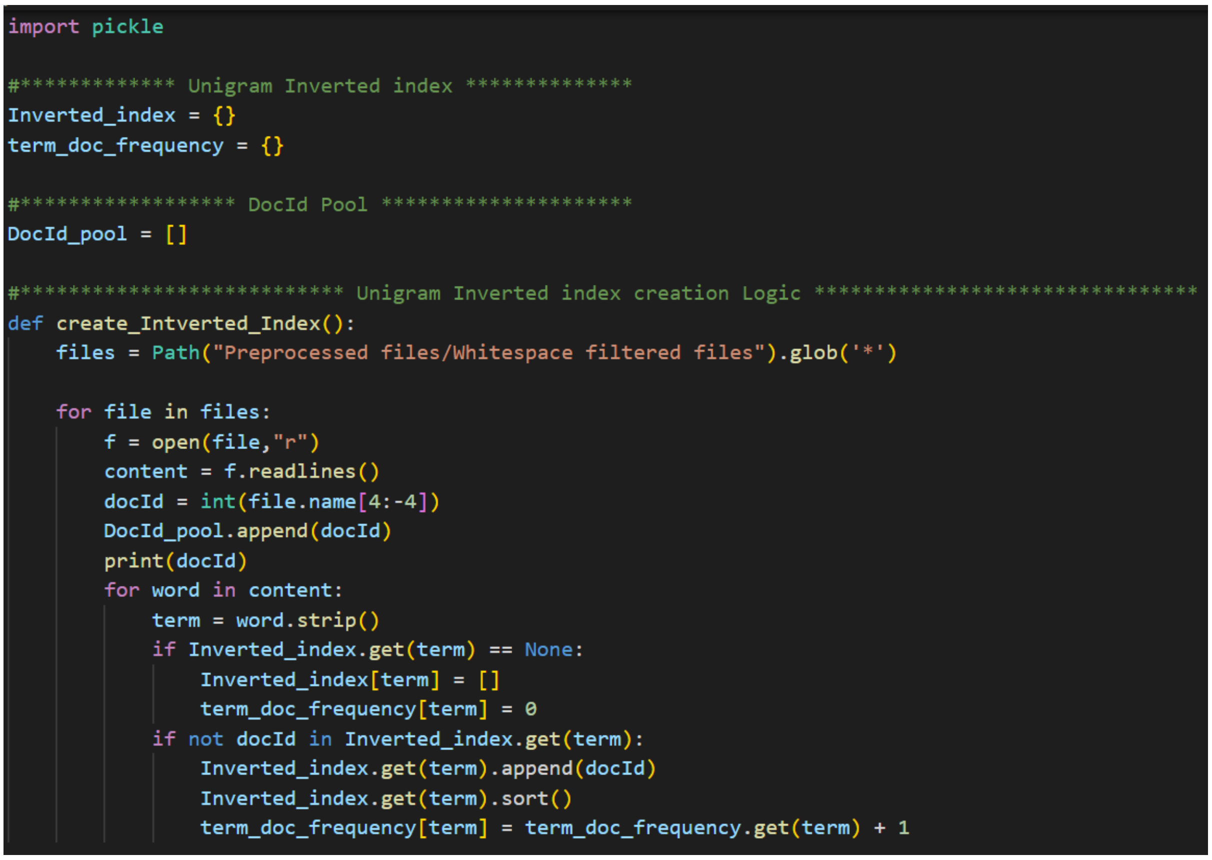
Methodologies:

Order of execution of the function



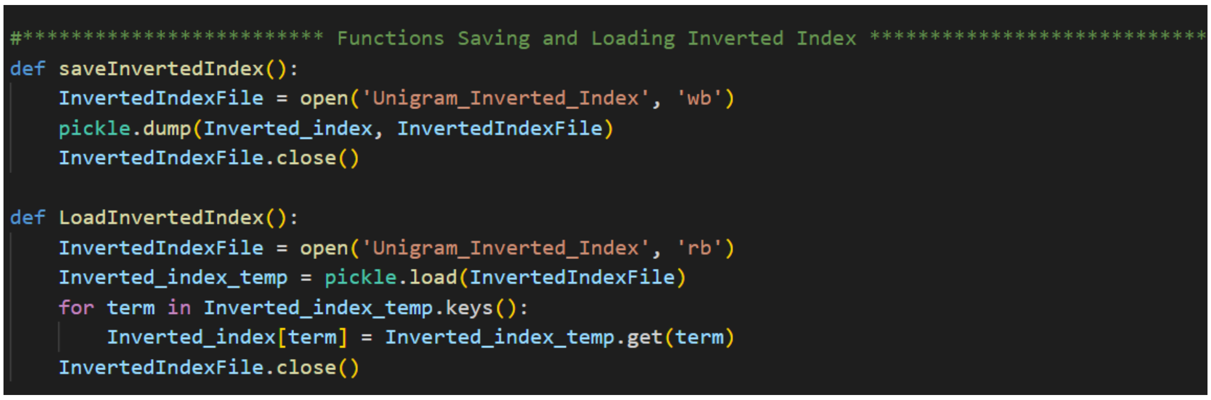
```python
#*********************** function calling *********************
create_Intverted_Index()
saveInvertedIndex()
LoadInvertedIndex()
Input_Output_Func()
```

This function create_Inverted_Index will create inverted index using dictionary. In this function we are traversing documents one by one only creating new term as key in dictionary if it is not already present as key and if we are creating key for the first time then we initialize the list and add document id in it else if the term already there in the dictionary as key then we will add document id in the list of document ids in the value portion of that term as key. Whenever I talked about dictionary means that inverted index.

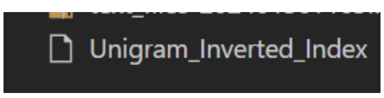


```python
import pickle

#************* Unigram Inverted index *************
Inverted_index = {}
term_doc_frequency = {}

#***************** DocId Pool ********************
DocId_pool = []

#*************************** Unigram Inverted index creation Logic ********************************
def create_Intverted_Index():
    files = Path("Preprocessed files/Whitespace filtered files").glob('*')

    for file in files:
        f = open(file,"r")
        content = f.readlines()
        docId = int(file.name[4:-4])
        DocId_pool.append(docId)
        print(docId)
        for word in content:
            term = word.strip()
            if Inverted_index.get(term) == None:
                Inverted_index[term] = []
                term_doc_frequency[term] = 0
            if not docId in Inverted_index.get(term):
                Inverted_index.get(term).append(docId)
                Inverted_index.get(term).sort()
                term_doc_frequency[term] = term_doc_frequency.get(term) + 1
```

Save inverted index Function for storing inverted index into byte format and Load inverted index function for loading stored byte format inverted index into python dictionary.

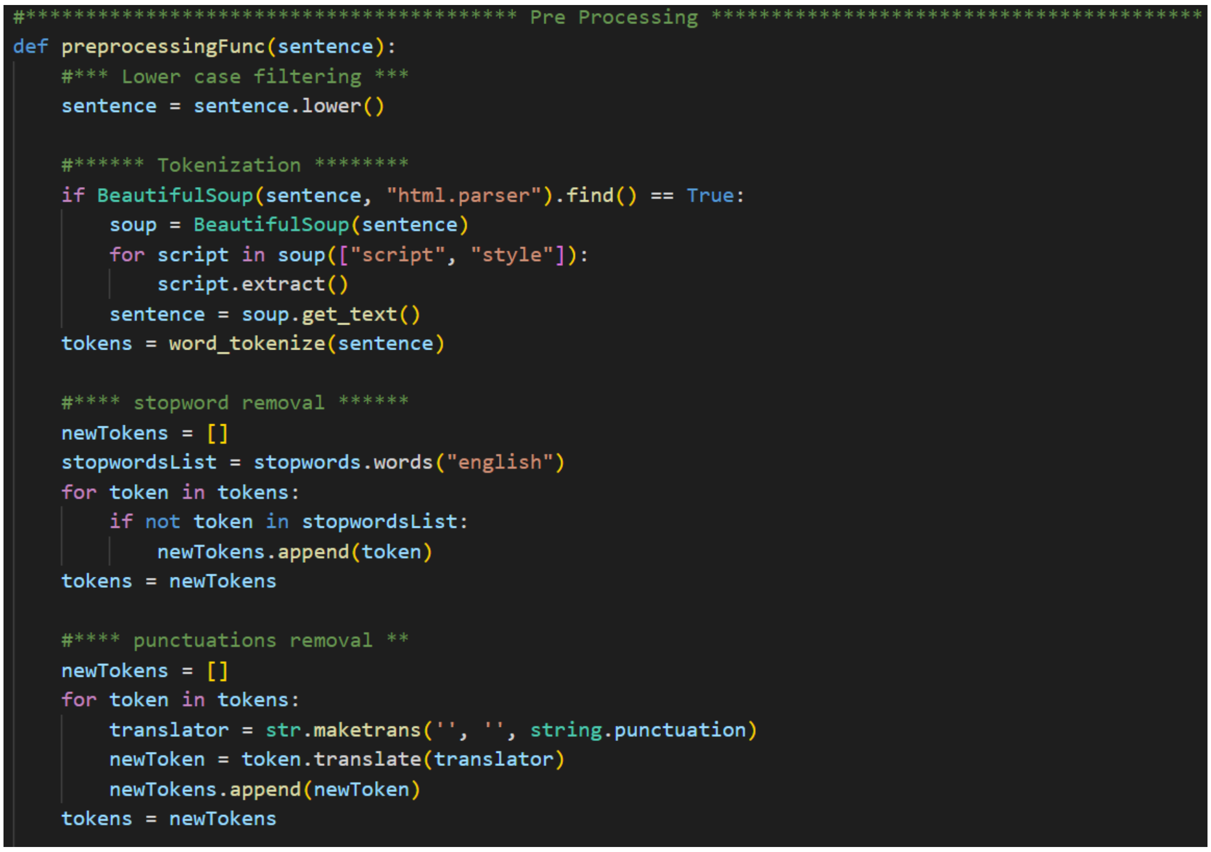


```python
#************************ Functions Saving and Loading Inverted Index ****************************
def saveInvertedIndex():
    InvertedIndexFile = open('Unigram_Inverted_Index', 'wb')
    pickle.dump(Inverted_index, InvertedIndexFile)
    InvertedIndexFile.close()

def LoadInvertedIndex():
    InvertedIndexFile = open('Unigram_Inverted_Index', 'rb')
    Inverted_index_temp = pickle.load(InvertedIndexFile)
    for term in Inverted_index_temp.keys():
        Inverted_index[term] = Inverted_index_temp.get(term)
    InvertedIndexFile.close()
```

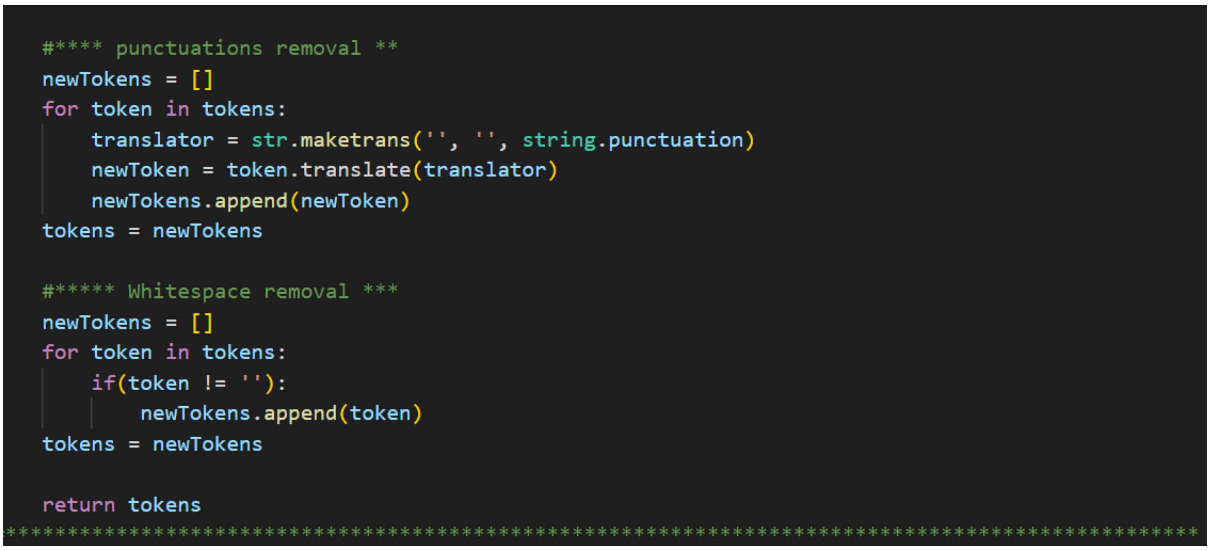


Unigram_Inverted_Index

In the below function we are doing preprocessing for the Query got through the input. We folled same steps for preprocessing as we performed in Question 1



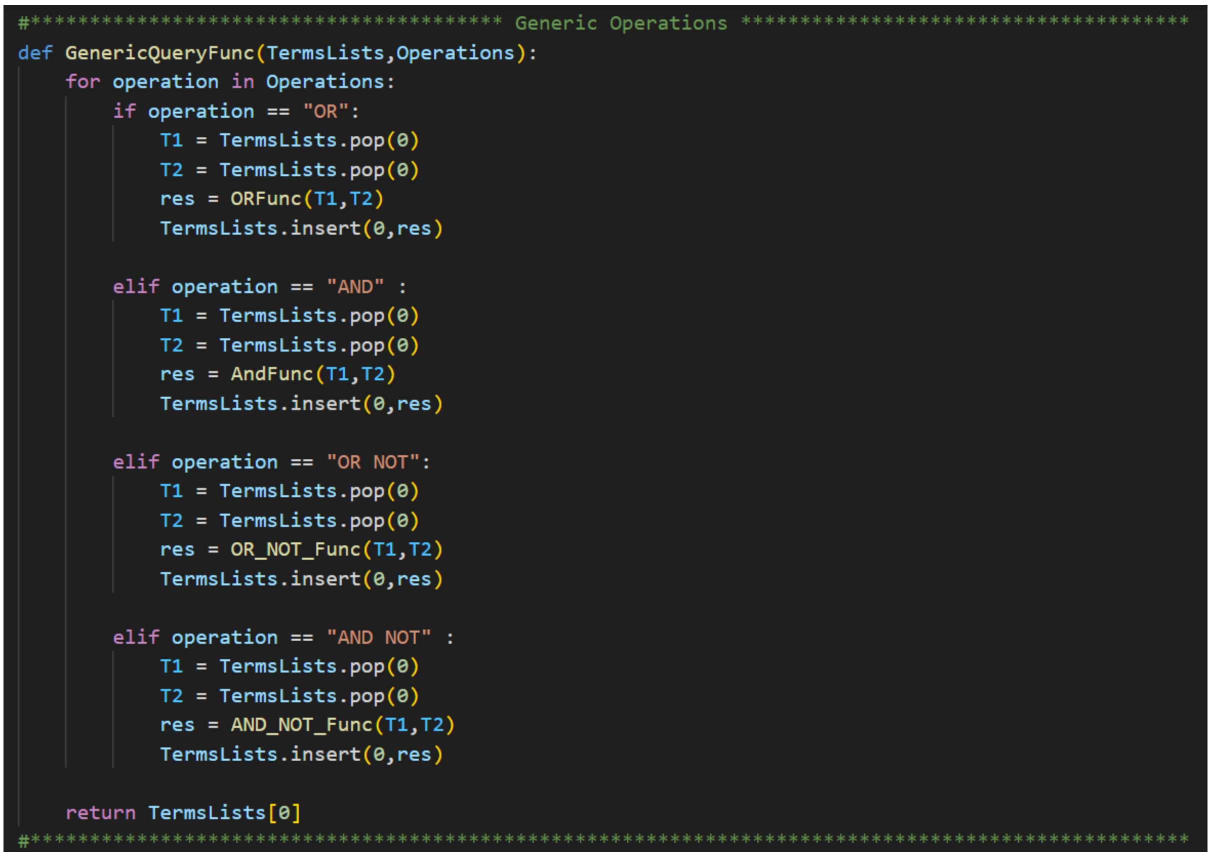
```
#****************************************** Pre Processing *******************************************
def preprocessingFunc(sentence):
    #*** Lower case filtering ***
    sentence = sentence.lower()

    #****** Tokenization ********
    if BeautifulSoup(sentence, "html.parser").find() == True:
        soup = BeautifulSoup(sentence)
        for script in soup(["script", "style"]):
            script.extract()
        sentence = soup.get_text()
    tokens = word_tokenize(sentence)

    #**** stopword removal ******
    newTokens = []
    stopwordsList = stopwords.words("english")
    for token in tokens:
        if not token in stopwordsList:
            newTokens.append(token)
    tokens = newTokens

    #**** punctuations removal **
    newTokens = []
    for token in tokens:
        translator = str.maketrans('', '', string.punctuation)
        newToken = token.translate(translator)
        newTokens.append(newToken)
    tokens = newTokens
```



```
    #**** punctuations removal **
    newTokens = []
    for token in tokens:
        translator = str.maketrans('', '', string.punctuation)
        newToken = token.translate(translator)
        newTokens.append(newToken)
    tokens = newTokens
```

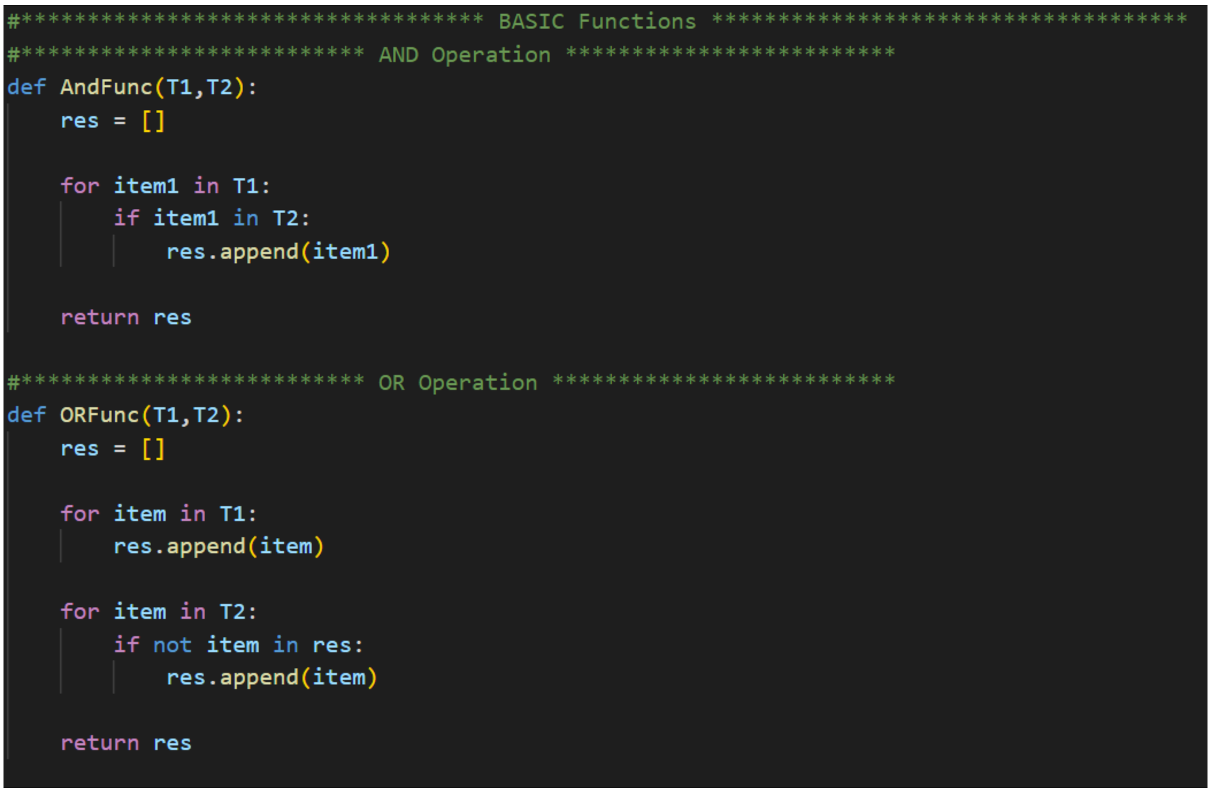
```
    #***** Whitespace removal ***
    newTokens = []
    for token in tokens:
        if(token != ''):
            newTokens.append(token)
    tokens = newTokens

    return tokens
****************************************************************************************************
```

This function GebericQueryFunc() is for processing Query's terms with operations given. In this we are processing terms in the Query in left to right fashion with respect to operations received as input. In this function we will get two lists one is list of document id's lists associated to each term mentioned in the Query in the same order of the terms in Query and second one is list of operations which is formed after splitting input operations by delimeter ','. Here we have handled cases for four types of operations AND , OR, AND NOT, OR NOT. In our function loop is looping for no of operations times for each operation we are performing operations on first two list of documents id after removing them list of lists of document id and after performing operations we are adding its result bask to first position of the list of lists of document id. At last returning left over list of doc id in list of lists of Document id

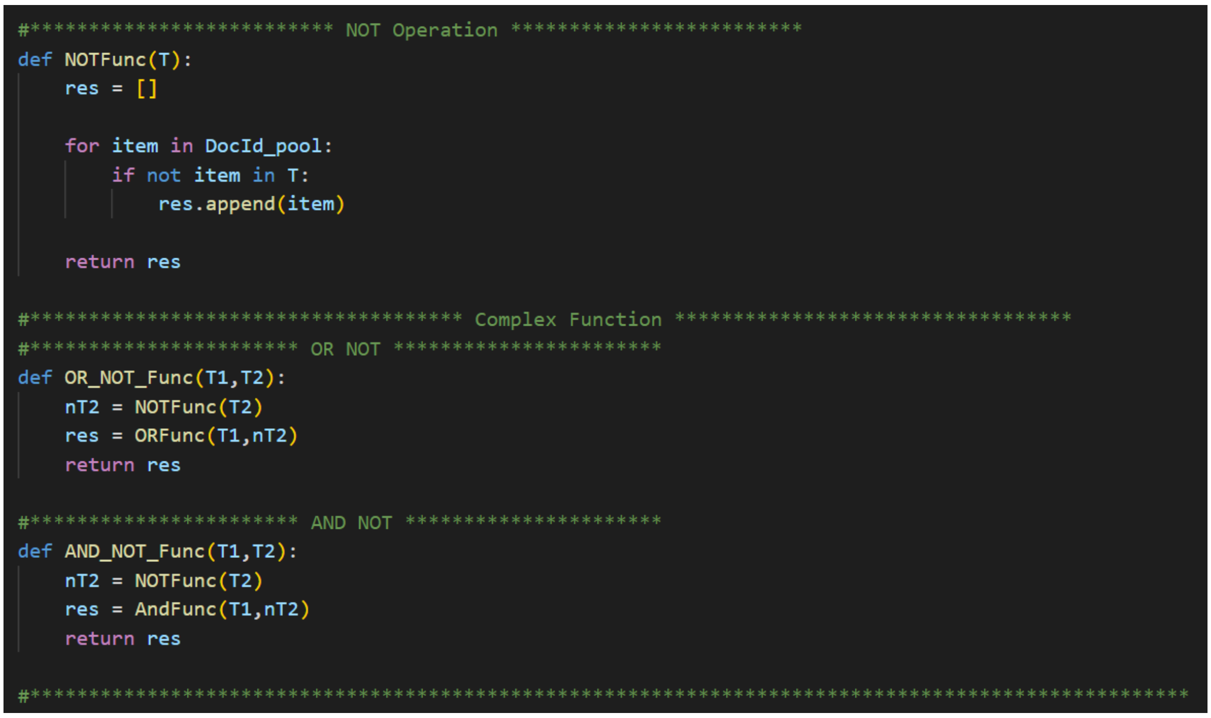


```python
#**************************************** Generic Operations **************************************
def GenericQueryFunc(TermsLists,Operations):
    for operation in Operations:
        if operation == "OR":
            T1 = TermsLists.pop(0)
            T2 = TermsLists.pop(0)
            res = ORFunc(T1,T2)
            TermsLists.insert(0,res)

        elif operation == "AND" :
            T1 = TermsLists.pop(0)
            T2 = TermsLists.pop(0)
            res = AndFunc(T1,T2)
            TermsLists.insert(0,res)

        elif operation == "OR NOT":
            T1 = TermsLists.pop(0)
            T2 = TermsLists.pop(0)
            res = OR_NOT_Func(T1,T2)
            TermsLists.insert(0,res)

        elif operation == "AND NOT" :
            T1 = TermsLists.pop(0)
            T2 = TermsLists.pop(0)
            res = AND_NOT_Func(T1,T2)
            TermsLists.insert(0,res)

    return TermsLists[0]
#***************************************************************************************************
```

These below are the basic functions that are binary operations like AND , OR , AND NOT , OR NOT. These functions take input two list of document Ids and perform corresponding operations.

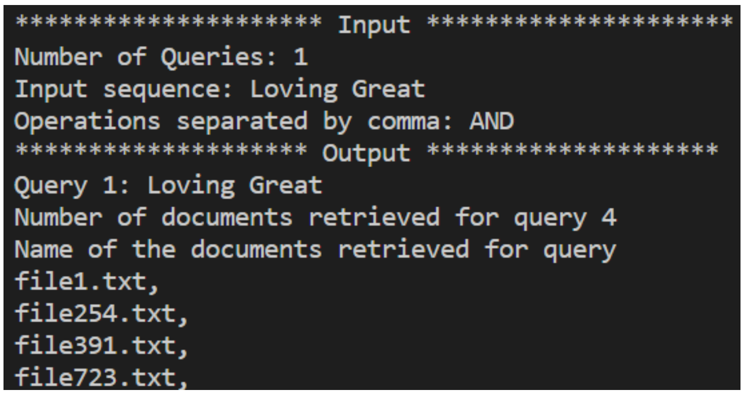


```python
#*********************************** BASIC Functions ************************************
#************************** AND Operation *************************
def AndFunc(T1,T2):
    res = []

    for item1 in T1:
        if item1 in T2:
            res.append(item1)

    return res

#************************** OR Operation **************************
def ORFunc(T1,T2):
    res = []

    for item in T1:
        res.append(item)

    for item in T2:
        if not item in res:
            res.append(item)

    return res
```

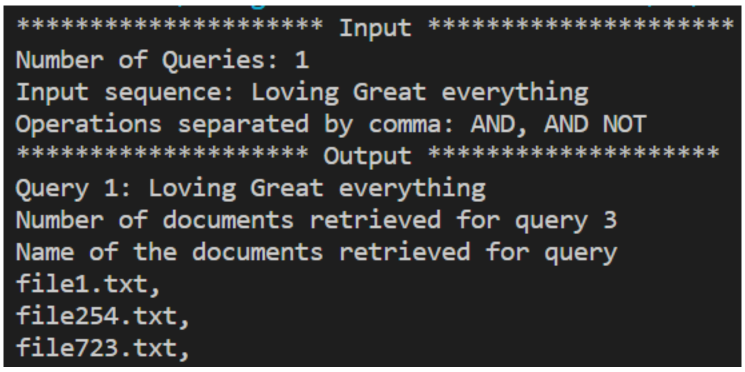


```
#************************** NOT Operation ************************
def NOTFunc(T):
    res = []

    for item in DocId_pool:
        if not item in T:
            res.append(item)

    return res

#************************************* Complex Function *********************************
#*********************** OR NOT **********************
def OR_NOT_Func(T1,T2):
    nT2 = NOTFunc(T2)
    res = ORFunc(T1,nT2)
    return res

#*********************** AND NOT *********************
def AND_NOT_Func(T1,T2):
    nT2 = NOTFunc(T2)
    res = AndFunc(T1,nT2)
    return res

#*************************************************************************************************
```

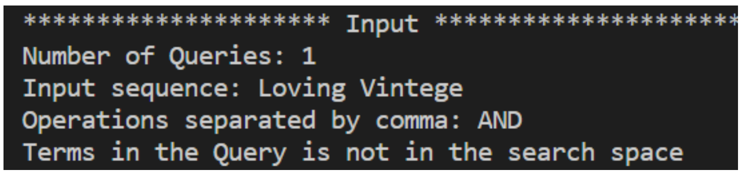
Results:



```
******************** Input ********************
Number of Queries: 1
Input sequence: Loving Great
Operations separated by comma: AND
******************** Output *******************
Query 1: Loving Great
Number of documents retrieved for query 4
Name of the documents retrieved for query
file1.txt,
file254.txt,
file391.txt,
file723.txt,
```



```
******************** Input ********************
Number of Queries: 1
Input sequence: Loving Great everything
Operations separated by comma: AND, AND NOT
******************** Output *******************
Query 1: Loving Great everything
Number of documents retrieved for query 3
Name of the documents retrieved for query
file1.txt,
file254.txt,
file723.txt,
```

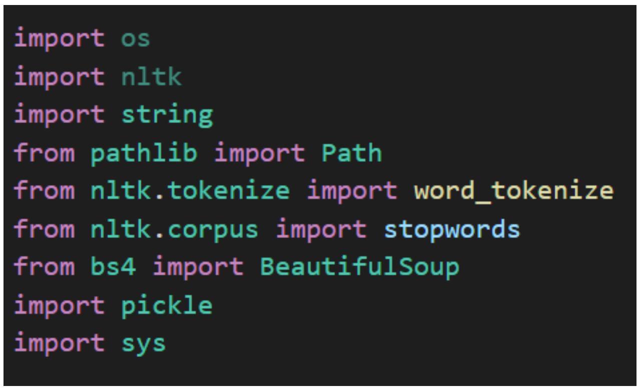


```
******************** Input ********************
Number of Queries: 1
Input sequence: Loving Vintege
Operations separated by comma: AND
Terms in the Query is not in the search space
```

Q3

Approach:

Libraries used:

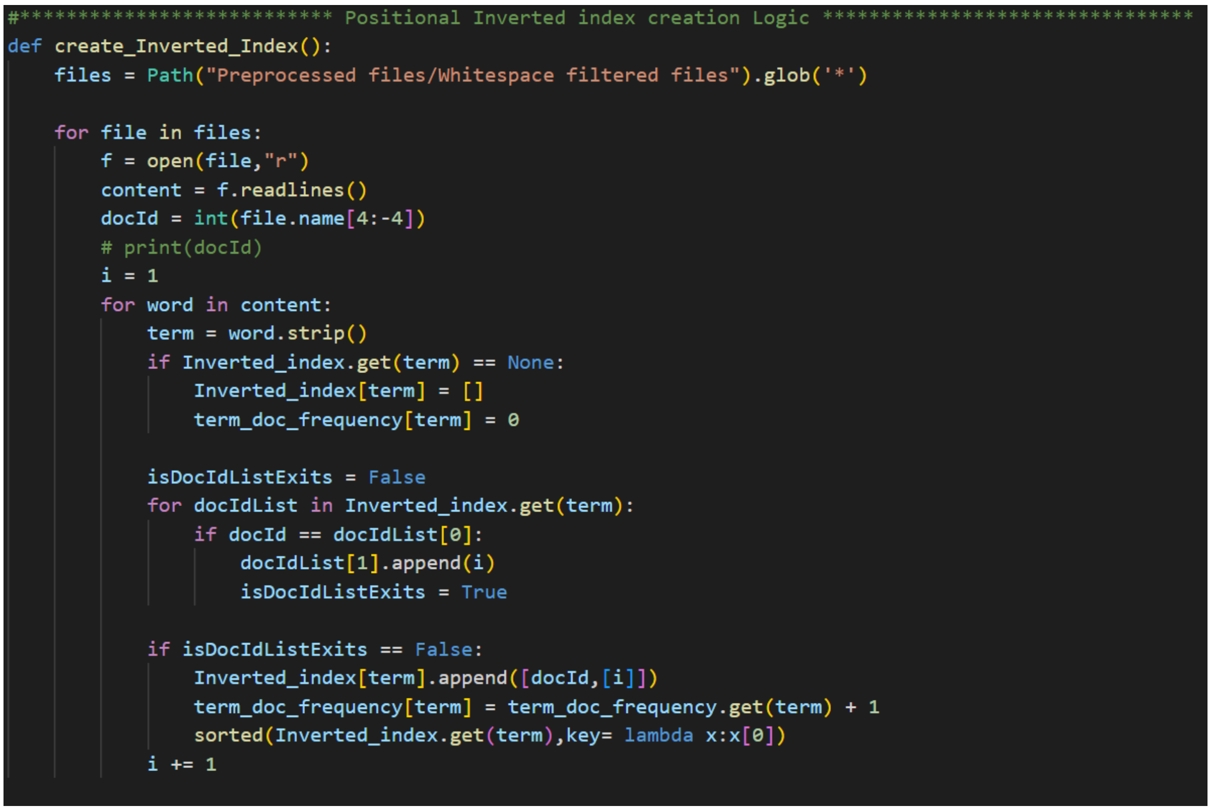


```
import os
import nltk
import string
from pathlib import Path
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from bs4 import BeautifulSoup
import pickle
import sys
```

In this we created inverted index same as Question 2 but in this we also maintained the list of positions of each term inside the each document. In this we used find contained documents to find which document are having the terms in query in the same sequence as of query. Our approach for finding the documents that are having same sequence of query and its is recursive . In this each take first term of the query and check in its corresponding documents whether the another terms are present in the same document or not in same sequence position if not then this document will not be in answer else this document will be in answer.

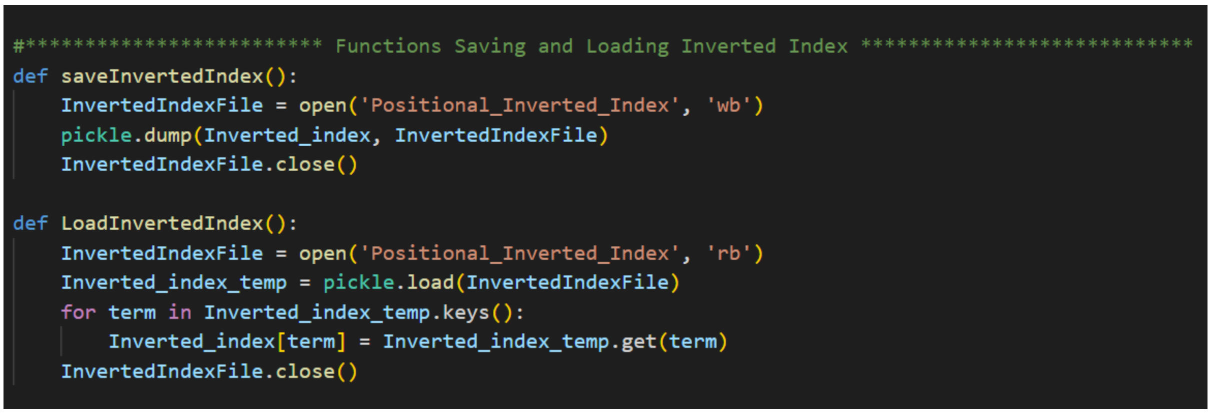
Methodologies:

This function below create the positional inverted index after reading all the files.

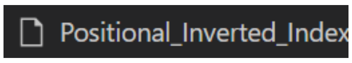


```
#************************** Positional Inverted index creation Logic ********************************
def create_Inverted_Index():
    files = Path("Preprocessed files/Whitespace filtered files").glob('*')

    for file in files:
        f = open(file,"r")
        content = f.readlines()
        docId = int(file.name[4:-4])
        # print(docId)
        i = 1
        for word in content:
            term = word.strip()
            if Inverted_index.get(term) == None:
                Inverted_index[term] = []
                term_doc_frequency[term] = 0

            isDocIdListExits = False
            for docIdList in Inverted_index.get(term):
                if docId == docIdList[0]:
                    docIdList[1].append(i)
                    isDocIdListExits = True

            if isDocIdListExits == False:
                Inverted_index[term].append([docId,[i]])
                term_doc_frequency[term] = term_doc_frequency.get(term) + 1
                sorted(Inverted_index.get(term),key= lambda x:x[0])
            i += 1
```

The function saveinvertedindex() is used to save positional inverted index into byte format and loadInvertedIndex() is used to load the byte format positional inverted index to python dictionary format positional inverted index.

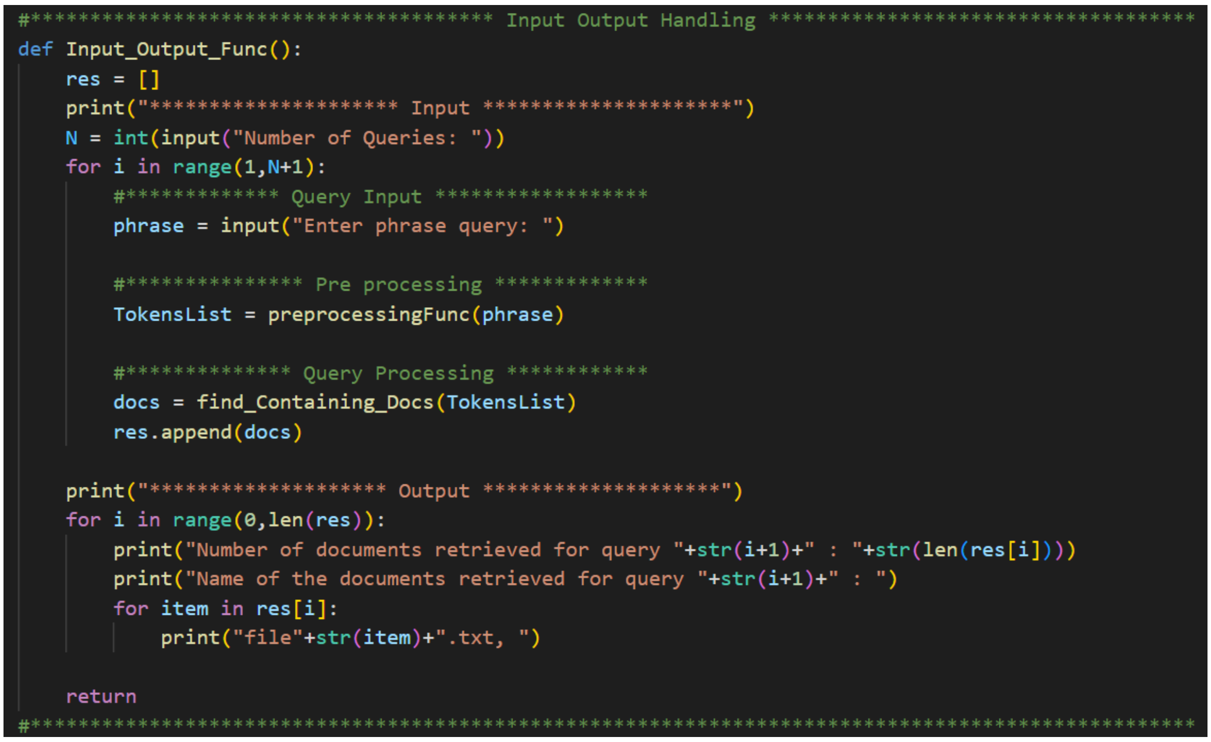


```python
#************************ Functions Saving and Loading Inverted Index ***************************
def saveInvertedIndex():
    InvertedIndexFile = open('Positional_Inverted_Index', 'wb')
    pickle.dump(Inverted_index, InvertedIndexFile)
    InvertedIndexFile.close()

def LoadInvertedIndex():
    InvertedIndexFile = open('Positional_Inverted_Index', 'rb')
    Inverted_index_temp = pickle.load(InvertedIndexFile)
    for term in Inverted_index_temp.keys():
        Inverted_index[term] = Inverted_index_temp.get(term)
    InvertedIndexFile.close()
```



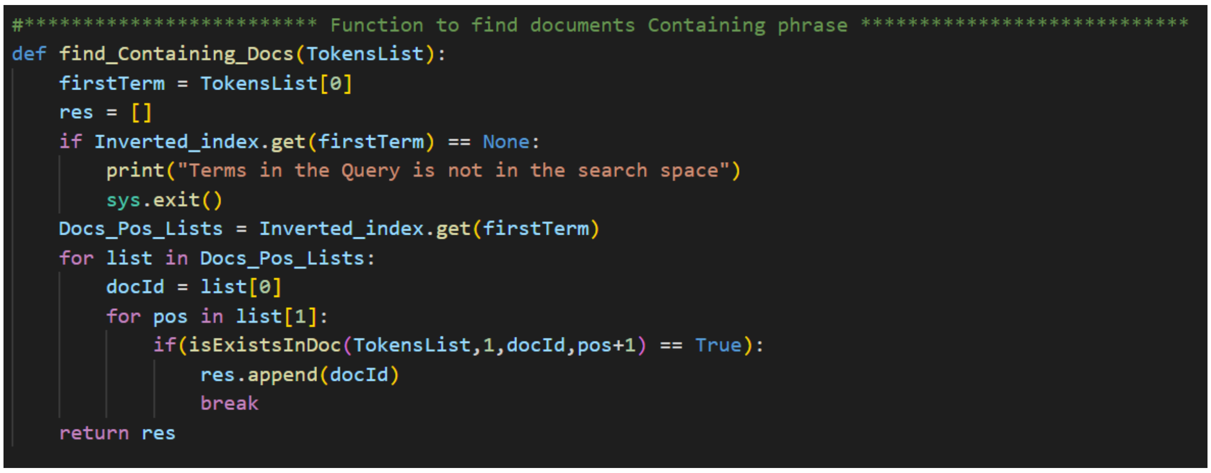
Positional_Inverted_Index

This Input_output_Func() takes input all the queries and process and print all the results of the queries.

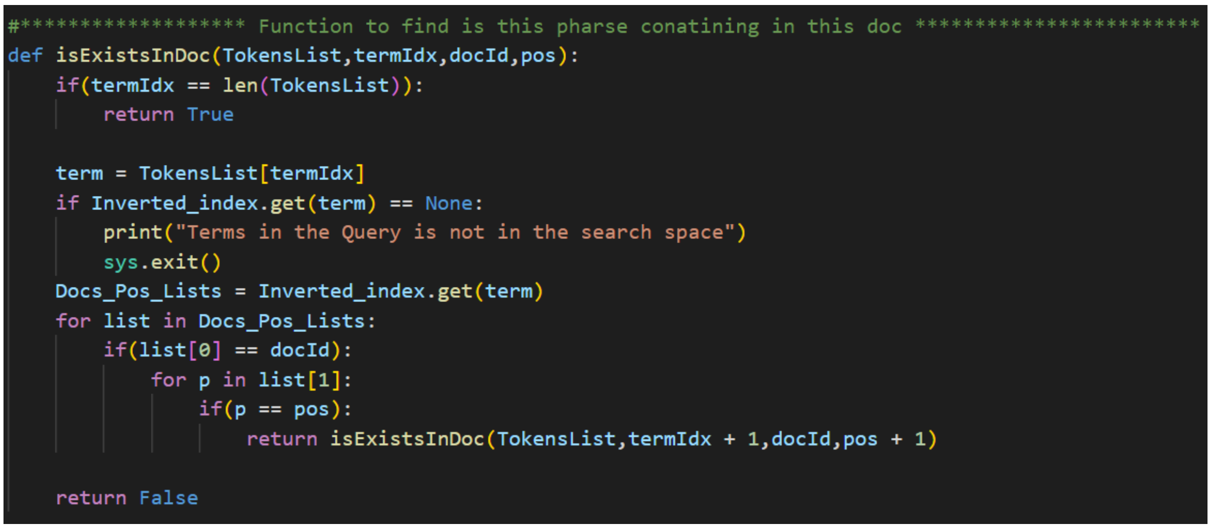


```python
#**************************************** Input Output Handling ************************************
def Input_Output_Func():
    res = []
    print("******************** Input ********************")
    N = int(input("Number of Queries: "))
    for i in range(1,N+1):
        #************* Query Input *****************
        phrase = input("Enter phrase query: ")

        #*************** Pre processing *************
        TokensList = preprocessingFunc(phrase)

        #************** Query Processing ***********
        docs = find_Containing_Docs(TokensList)
        res.append(docs)

    print("******************* Output *******************")
    for i in range(0,len(res)):
        print("Number of documents retrieved for query "+str(i+1)+" : "+str(len(res[i])))
        print("Name of the documents retrieved for query "+str(i+1)+" : ")
        for item in res[i]:
            print("file"+str(item)+".txt, ")

    return
#**************************************************************************************************
```

These below two helper functions are used to find the documents that are having query terms inputted by user in the same sequence as mentioned in query.

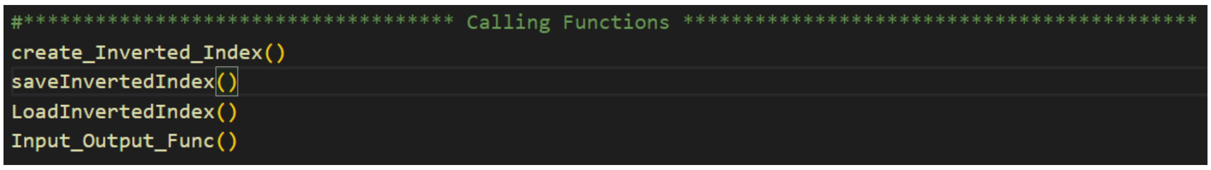


```python
#************************* Function to find documents Containing phrase ****************************
def find_Containing_Docs(TokensList):
    firstTerm = TokensList[0]
    res = []
    if Inverted_index.get(firstTerm) == None:
        print("Terms in the Query is not in the search space")
        sys.exit()
    Docs_Pos_Lists = Inverted_index.get(firstTerm)
    for list in Docs_Pos_Lists:
        docId = list[0]
        for pos in list[1]:
            if(isExistsInDoc(TokensList,1,docId,pos+1) == True):
                res.append(docId)
                break
    return res
```



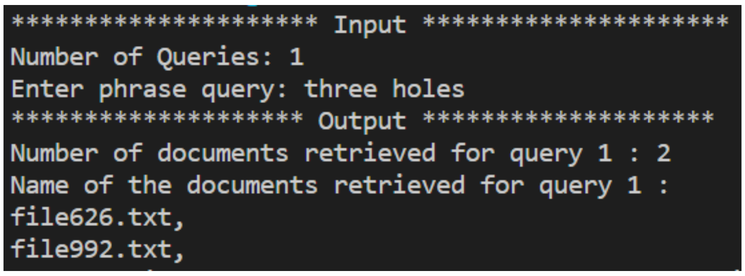
```python
#****************** Function to find is this pharse conatining in this doc ************************
def isExistsInDoc(TokensList,termIdx,docId,pos):
    if(termIdx == len(TokensList)):
        return True

    term = TokensList[termIdx]
    if Inverted_index.get(term) == None:
        print("Terms in the Query is not in the search space")
        sys.exit()
    Docs_Pos_Lists = Inverted_index.get(term)
    for list in Docs_Pos_Lists:
        if(list[0] == docId):
            for p in list[1]:
                if(p == pos):
                    return isExistsInDoc(TokensList,termIdx + 1,docId,pos + 1)

    return False
```

Order of function execution



```python
#************************************* Calling Functions *********************************************
create_Inverted_Index()
saveInvertedIndex()
LoadInvertedIndex()
Input_Output_Func()
```

Results:



```
******************** Input *********************
Number of Queries: 1
Enter phrase query: three holes
******************* Output ********************
Number of documents retrieved for query 1 : 2
Name of the documents retrieved for query 1 :
file626.txt,
file992.txt,
```