Assignment: 1

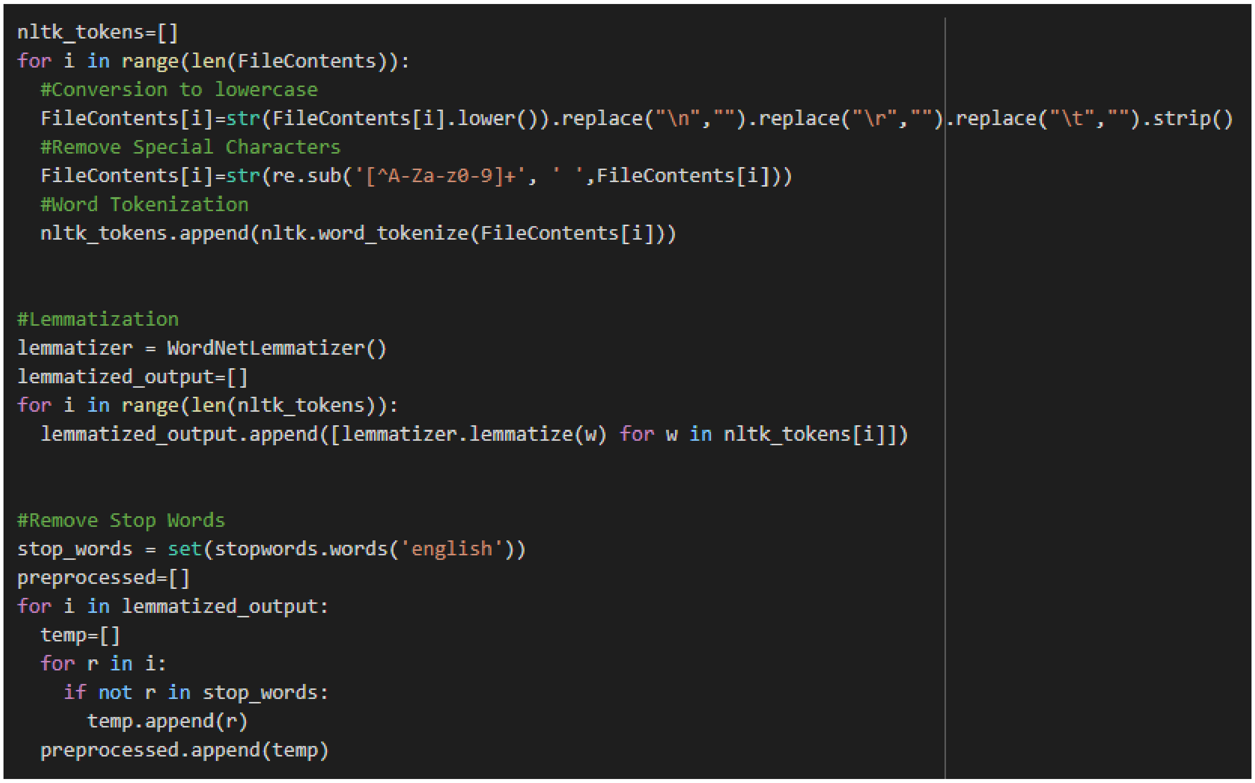
Q-1

- We have extracted all the file paths in ‘alldirs’ and after that, we have sorted ‘alldirs’ by using ‘alldirs.sort()’.
- After sorting ‘alldirs’, the content of each file is stored in FileContent i.e FileContent[0] stores the content of the first file.

A. Preprocessing:

We have done the following preprocessing steps

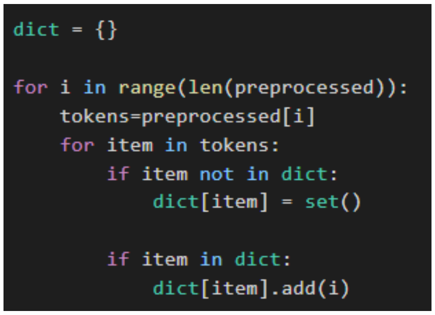
a. **Conversion to lowercase**
b. **Removal of special characters**
c. **Word Tokenization**
d. **Lemmatization**
e. **Remove Stopwords**



```
nltk_tokens=[]
for i in range(len(FileContents)):
  #Conversion to lowercase
  FileContents[i]=str(FileContents[i].lower()).replace("\n","").replace("\r","").replace("\t","").strip()
  #Remove Special Characters
  FileContents[i]=str(re.sub('[^A-Za-z0-9]+', ' ',FileContents[i]))
  #Word Tokenization
  nltk_tokens.append(nltk.word_tokenize(FileContents[i]))


#Lemmatization
lemmatizer = WordNetLemmatizer()
lemmatized_output=[]
for i in range(len(nltk_tokens)):
  lemmatized_output.append([lemmatizer.lemmatize(w) for w in nltk_tokens[i]])


#Remove Stop Words
stop_words = set(stopwords.words('english'))
preprocessed=[]
for i in lemmatized_output:
  temp=[]
  for r in i:
    if not r in stop_words:
      temp.append(r)
  preprocessed.append(temp)
```

B. Implementing Unigram Inverted Index Data Structure

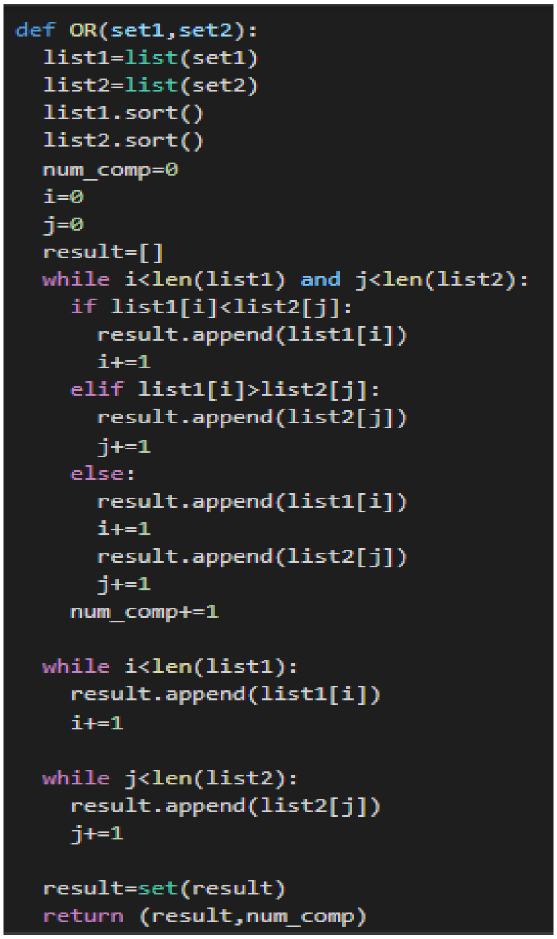
We have used a dictionary to map tokens with the document sets they are present in.



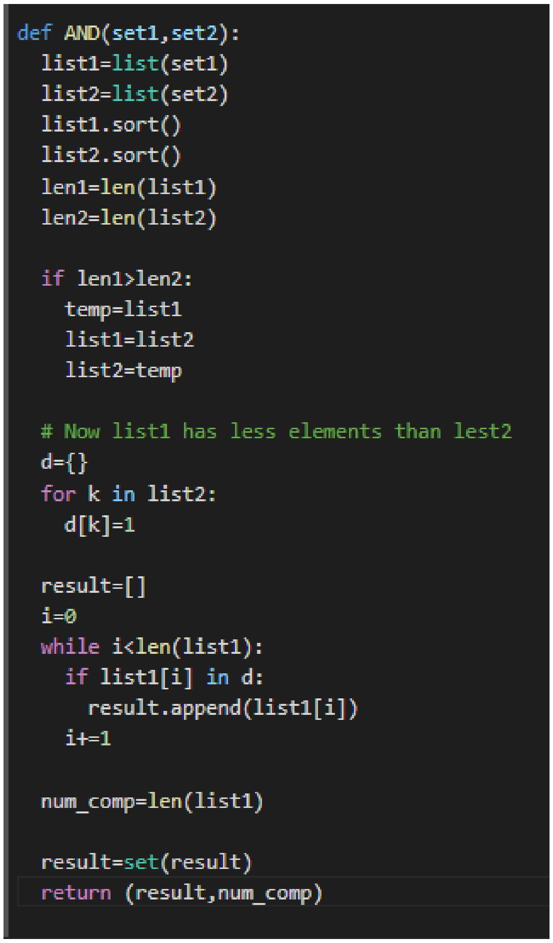
```
dict = {}

for i in range(len(preprocessed)):
    tokens=preprocessed[i]
    for item in tokens:
        if item not in dict:
            dict[item] = set()

        if item in dict:
            dict[item].add(i)
```

C. Implementation of OR,AND, AND NOT, OR NOT

We have implemented OR by using merge algorithm. We also have counted the number of comparisons made

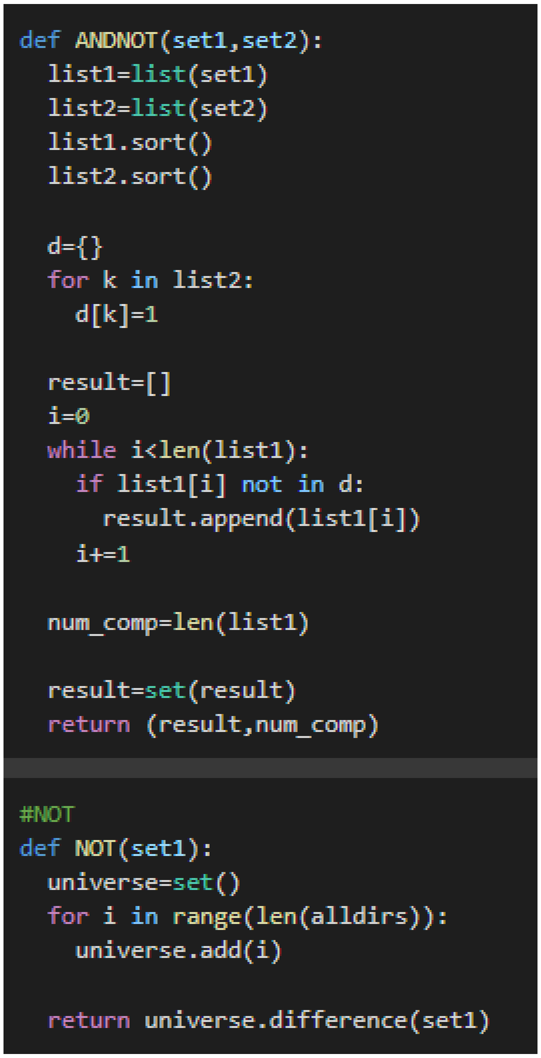


```
def OR(set1,set2):
  list1=list(set1)
  list2=list(set2)
  list1.sort()
  list2.sort()
  num_comp=0
  i=0
  j=0
  result=[]
  while i<len(list1) and j<len(list2):
    if list1[i]<list2[j]:
      result.append(list1[i])
      i+=1
    elif list1[i]>list2[j]:
      result.append(list2[j])
      j+=1
    else:
      result.append(list1[i])
      i+=1
      result.append(list2[j])
      j+=1
    num_comp+=1

  while i<len(list1):
    result.append(list1[i])
    i+=1

  while j<len(list2):
    result.append(list2[j])
    j+=1

  result=set(result)
  return (result,num_comp)
```



```
def AND(set1,set2):
  list1=list(set1)
  list2=list(set2)
  list1.sort()
  list2.sort()
  len1=len(list1)
  len2=len(list2)

  if len1>len2:
    temp=list1
    list1=list2
    list2=temp

  # Now list1 has less elements than lest2
  d={}
  for k in list2:
    d[k]=1

  result=[]
  i=0
  while i<len(list1):
    if list1[i] in d:
      result.append(list1[i])
    i+=1

  num_comp=len(list1)

  result=set(result)
  return (result,num_comp)
```

We have implemented AND by making a dictionary of the larger set of documents and checking whether any element of the smaller set is present in the larger set



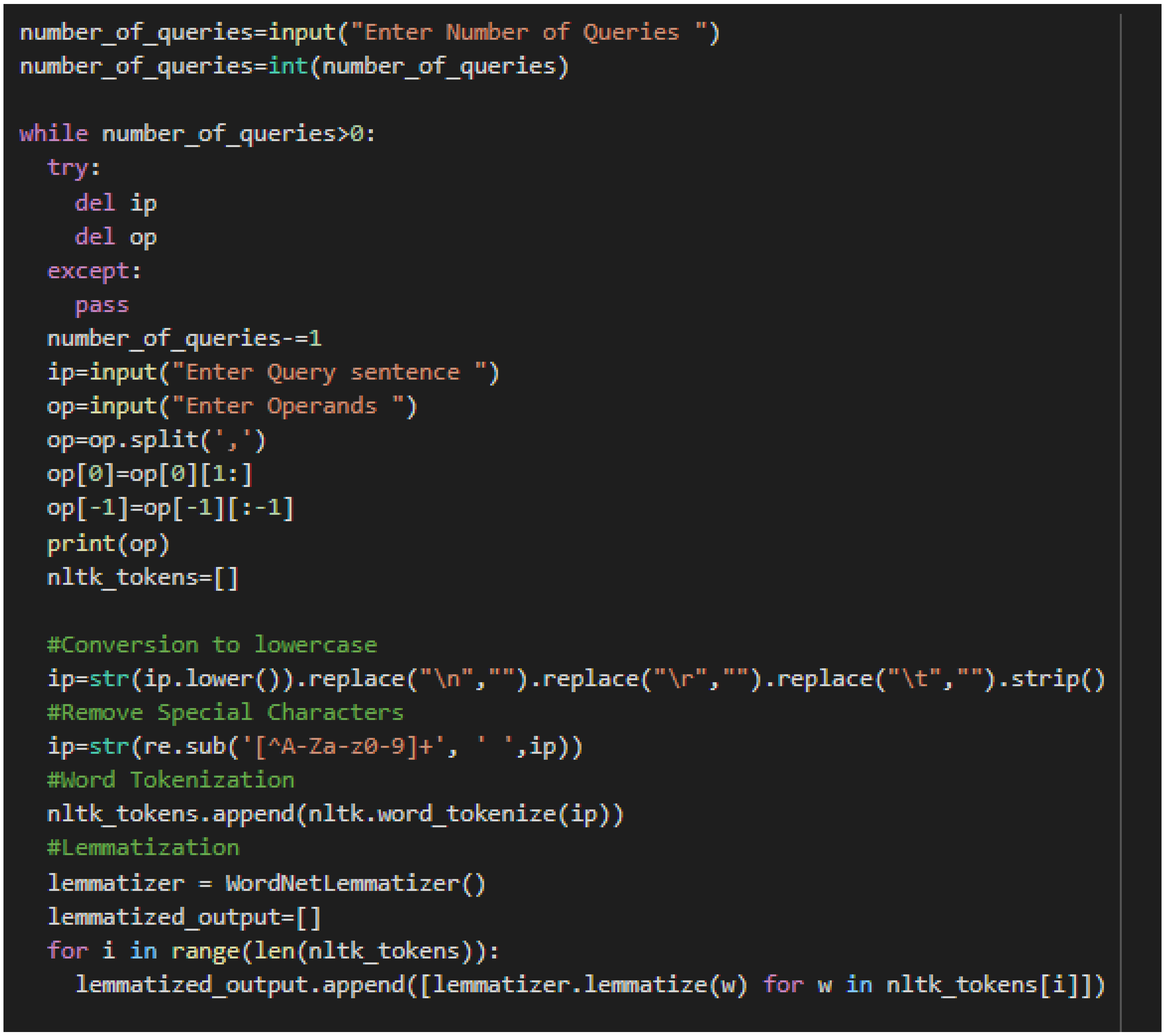
```
def ANDNOT(set1,set2):
  list1=list(set1)
  list2=list(set2)
  list1.sort()
  list2.sort()

  d={}
  for k in list2:
    d[k]=1

  result=[]
  i=0
  while i<len(list1):
    if list1[i] not in d:
      result.append(list1[i])
    i+=1

  num_comp=len(list1)

  result=set(result)
  return (result,num_comp)
```

```
#NOT
def NOT(set1):
  universe=set()
  for i in range(len(alldirs)):
    universe.add(i)

  return universe.difference(set1)
```

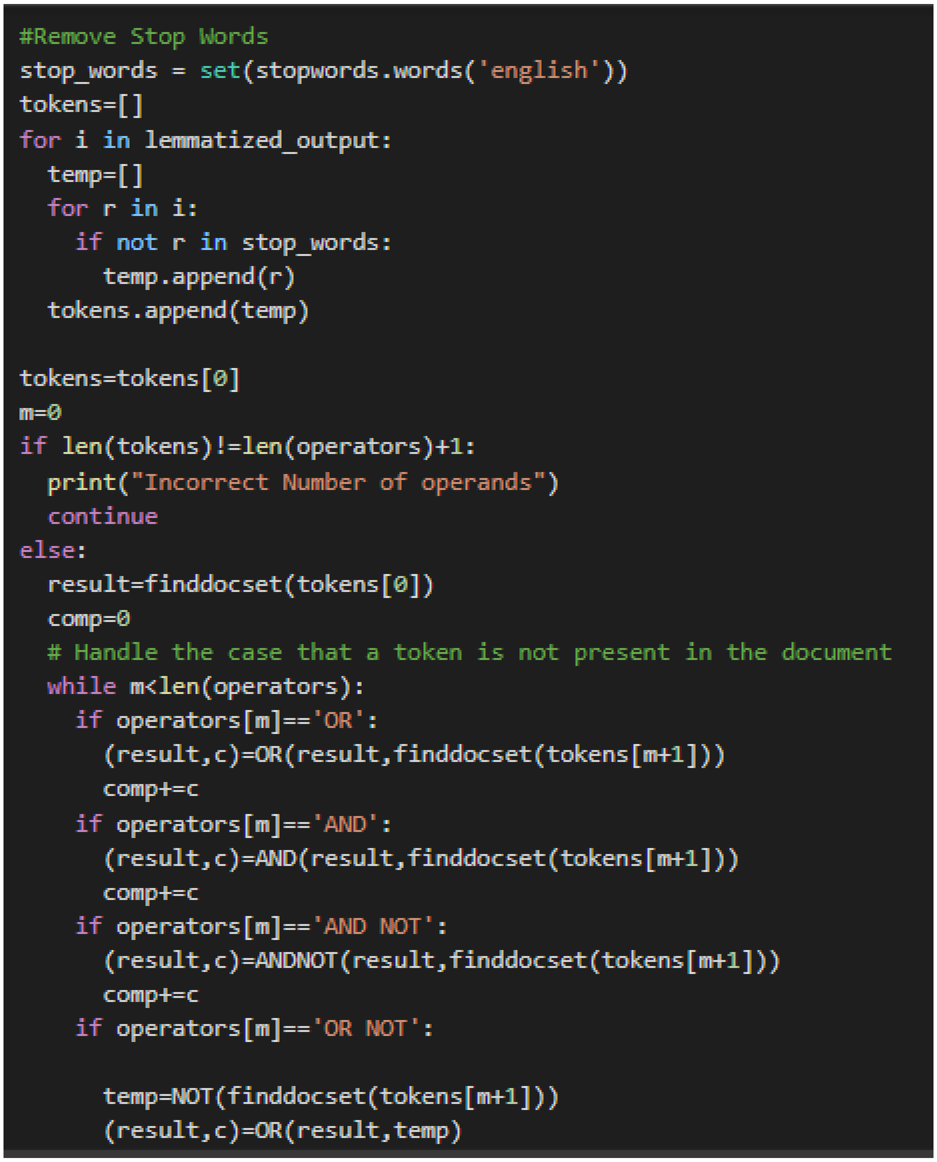
We have implemented OR NOT by taking NOT and then performing OR operation.

We have implemented AND NOT by writing the code for set difference from scratch

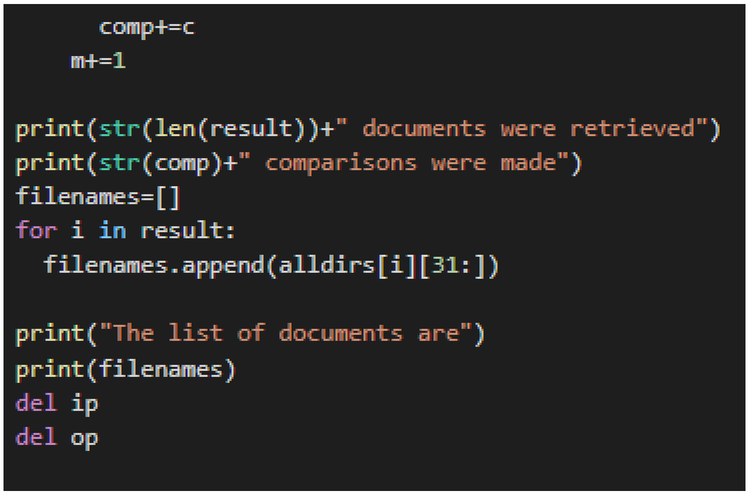
D. Providing Support for the mentioned input format



```
number_of_queries=input("Enter Number of Queries ")
number_of_queries=int(number_of_queries)

while number_of_queries>0:
  try:
    del ip
    del op
  except:
    pass
  number_of_queries-=1
  ip=input("Enter Query sentence ")
  op=input("Enter Operands ")
  op=op.split(',')
  op[0]=op[0][1:]
  op[-1]=op[-1][:-1]
  print(op)
  nltk_tokens=[]

  #Conversion to lowercase
  ip=str(ip.lower()).replace("\n","").replace("\r","").replace("\t","").strip()
  #Remove Special Characters
  ip=str(re.sub('[^A-Za-z0-9]+', ' ',ip))
  #Word Tokenization
  nltk_tokens.append(nltk.word_tokenize(ip))
  #Lemmatization
  lemmatizer = WordNetLemmatizer()
  lemmatized_output=[]
  for i in range(len(nltk_tokens)):
    lemmatized_output.append([lemmatizer.lemmatize(w) for w in nltk_tokens[i]])
```

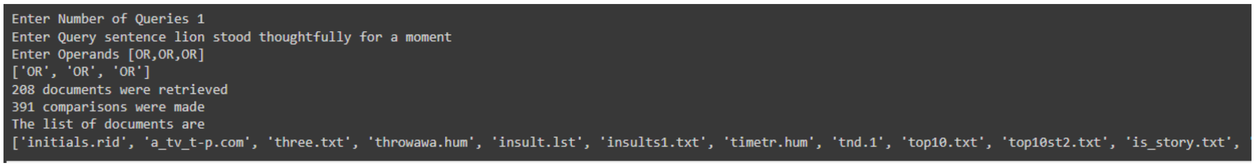


```python
#Remove Stop Words
stop_words = set(stopwords.words('english'))
tokens=[]
for i in lemmatized_output:
  temp=[]
  for r in i:
    if not r in stop_words:
      temp.append(r)
  tokens.append(temp)

tokens=tokens[0]
m=0
if len(tokens)!=len(operators)+1:
  print("Incorrect Number of operands")
  continue
else:
  result=finddocset(tokens[0])
  comp=0
  # Handle the case that a token is not present in the document
  while m<len(operators):
    if operators[m]=='OR':
      (result,c)=OR(result,finddocset(tokens[m+1]))
      comp+=c
    if operators[m]=='AND':
      (result,c)=AND(result,finddocset(tokens[m+1]))
      comp+=c
    if operators[m]=='AND NOT':
      (result,c)=ANDNOT(result,finddocset(tokens[m+1]))
      comp+=c
    if operators[m]=='OR NOT':

      temp=NOT(finddocset(tokens[m+1]))
      (result,c)=OR(result,temp)
```



```python
      comp+=c
    m+=1

print(str(len(result))+" documents were retrieved")
print(str(comp)+" comparisons were made")
filenames=[]
for i in result:
  filenames.append(alldirs[i][31:])

print("The list of documents are")
print(filenames)
del ip
del op
```

Sample Output :



```
Enter Number of Queries 1
Enter Query sentence lion stood thoughtfully for a moment
Enter Operands [OR,OR,OR]
['OR', 'OR', 'OR']
208 documents were retrieved
391 comparisons were made
The list of documents are
['initials.rid', 'a_tv_t-p.com', 'three.txt', 'throwawa.hum', 'insult.lst', 'insults1.txt', 'timetr.hum', 'tnd.1', 'top10.txt', 'top10st2.txt', 'is_story.txt',
```

Q-2

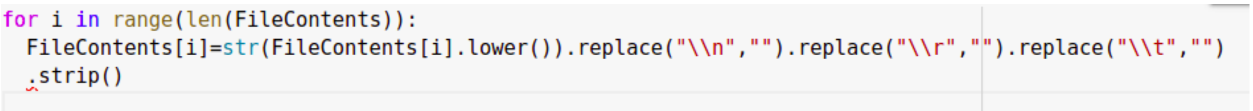
- We have extracted all the file paths in ‘alldirs’ and after that, we have sorted ‘alldirs’ by using ‘alldirs.sort()’.
- After sorting ‘alldirs’, the content of each file is stored in FileContent i.e FileContent[0] stores the content of the first file.

(a) Preprocessing:

Before creating the positional index data structure, we have done some preprocessing. The steps that we followed during preprocessing are:

(i) Converting text to lower case and removing the blank space:

We have converted the text into the lower case by using the following code:

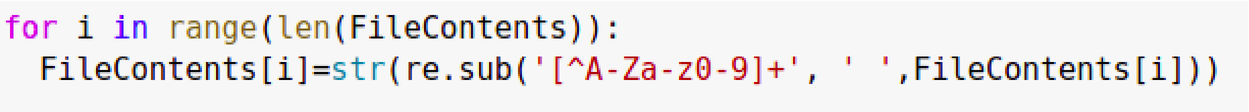


```python
for i in range(len(FileContents)):
  FileContents[i]=str(FileContents[i].lower()).replace("\\n","").replace("\\r","").replace("\\t","")
  .strip()
```

Also, we have removed any leading and trailing blank spaces by using ‘strip()’

(ii) Removing Special Characters (punctuation marks etc):

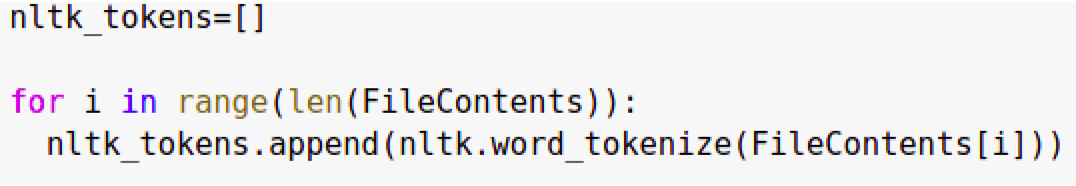
We have used regex for removing the special characters from our file contents. The code for the same is:



```python
for i in range(len(FileContents)):
  FileContents[i]=str(re.sub('[^A-Za-z0-9]+', ' ',FileContents[i]))
```

(iii) Performed Word Tokenization:

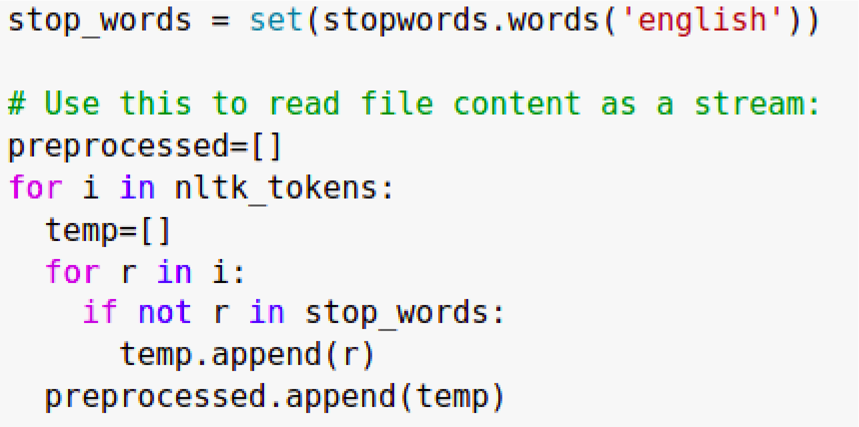
After that, we have performed the word tokenization and stored the tokens of each file into a list. So after this step, we finally have ‘nltk_tokens’ list, which is a list of token’s list of the file. The code for the same is:



```
nltk_tokens=[]

for i in range(len(FileContents)):
  nltk_tokens.append(nltk.word_tokenize(FileContents[i]))
```

(iv) Removing StopWords

After converting each file content into tokens, the next step is to remove the stopwords. We have downloaded the ‘nltk’ stopwords. The code for the same is:



```
stop_words = set(stopwords.words('english'))

# Use this to read file content as a stream:
preprocessed=[]
for i in nltk_tokens:
  temp=[]
  for r in i:
    if not r in stop_words:
      temp.append(r)
  preprocessed.append(temp)
```

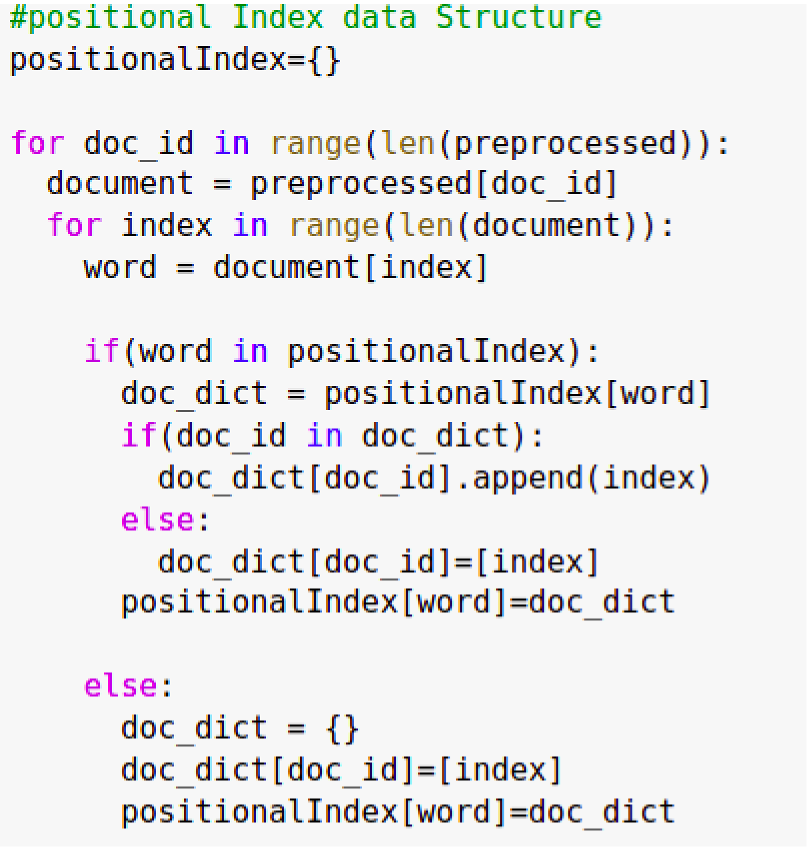
(b) Implementing the Positional Index Data Structure:

Now to resolve the phrase queries, we have implemented the Positional Index Data Structure.

We have created a ‘positionalIndex’ dictionary which maps a word with another dictionary. E.g for word ‘study’, positionalIndex[‘word’] = another dictionary (which maps document id to the indices list)

So the inner dictionary basically stores the list of indices of the positions where word (e.g ‘study’) has occurred in that particular document.

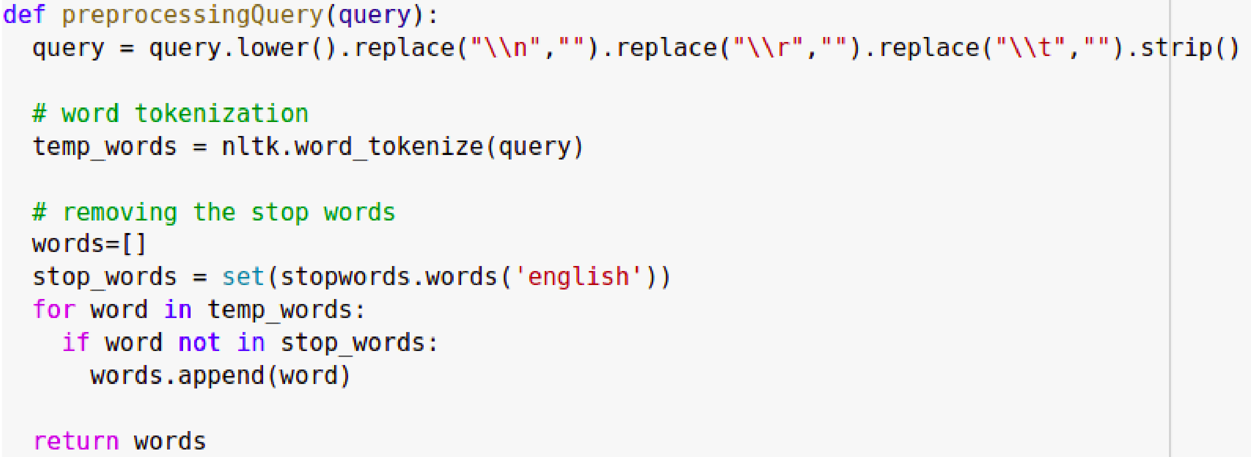
The code for the positional index data structure is as follows:



```
#positional Index data Structure
positionalIndex={}

for doc_id in range(len(preprocessed)):
  document = preprocessed[doc_id]
  for index in range(len(document)):
    word = document[index]

    if(word in positionalIndex):
      doc_dict = positionalIndex[word]
      if(doc_id in doc_dict):
        doc_dict[doc_id].append(index)
      else:
        doc_dict[doc_id]=[index]
      positionalIndex[word]=doc_dict

    else:
      doc_dict = {}
      doc_dict[doc_id]=[index]
      positionalIndex[word]=doc_dict
```

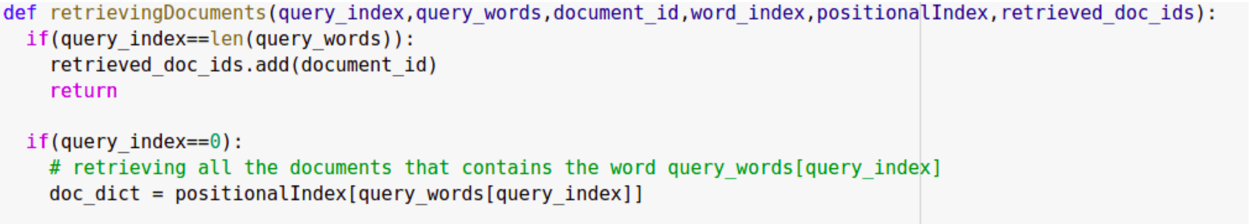
(c) Providing support for the phrase queries:

We have taken input the phrase queries in the form of list. Our first step is to preprocess the phrase queries in the same way as we have preprocessed each file contents (the same step as mentioned above are followed for phrase query preprocessing)

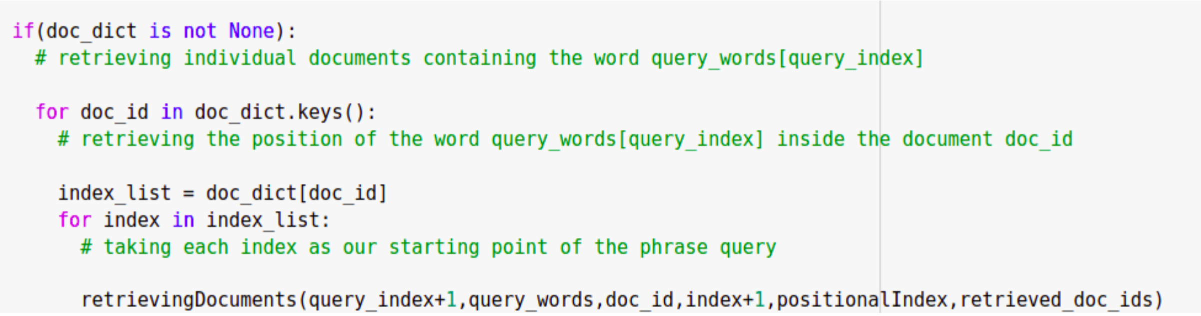


```python
def preprocessingQuery(query):
  query = query.lower().replace("\\n","").replace("\\r","").replace("\\t","").strip()

  # word tokenization
  temp_words = nltk.word_tokenize(query)

  # removing the stop words
  words=[]
  stop_words = set(stopwords.words('english'))
  for word in temp_words:
    if word not in stop_words:
      words.append(word)

  return words
```

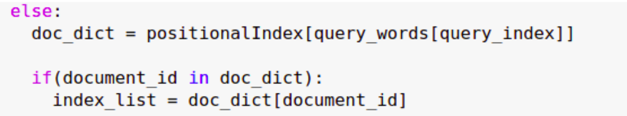
We have also defined a function for retrieving the document ids which contains the phrase queries, the function is:



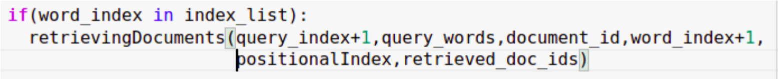
```python
def retrievingDocuments(query_index,query_words,document_id,word_index,positionalIndex,retrieved_doc_ids):
  if(query_index==len(query_words)):
    retrieved_doc_ids.add(document_id)
    return

  if(query_index==0):
    # retrieving all the documents that contains the word query_words[query_index]
    doc_dict = positionalIndex[query_words[query_index]]
```



```python
    if(doc_dict is not None):
      # retrieving individual documents containing the word query_words[query_index]

      for doc_id in doc_dict.keys():
        # retrieving the position of the word query_words[query_index] inside the document doc_id

        index_list = doc_dict[doc_id]
        for index in index_list:
          # taking each index as our starting point of the phrase query

          retrievingDocuments(query_index+1,query_words,doc_id,index+1,positionalIndex,retrieved_doc_ids)
```

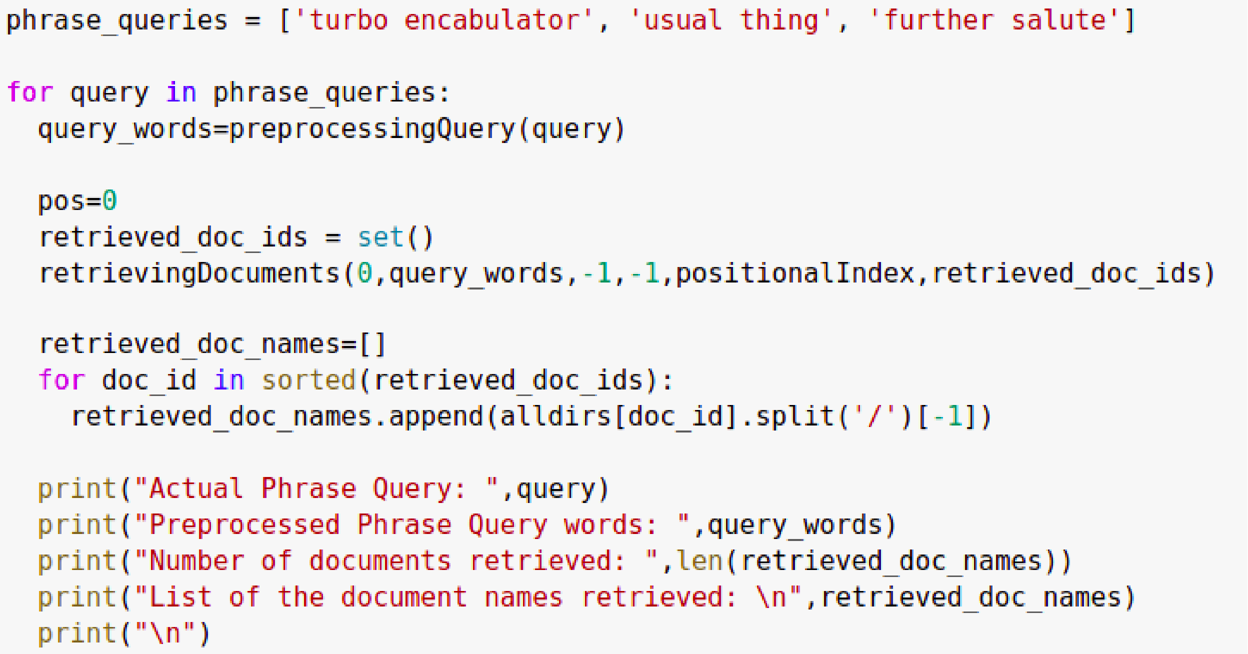


```python
  else:
    doc_dict = positionalIndex[query_words[query_index]]

    if(document_id in doc_dict):
      index_list = doc_dict[document_id]
```



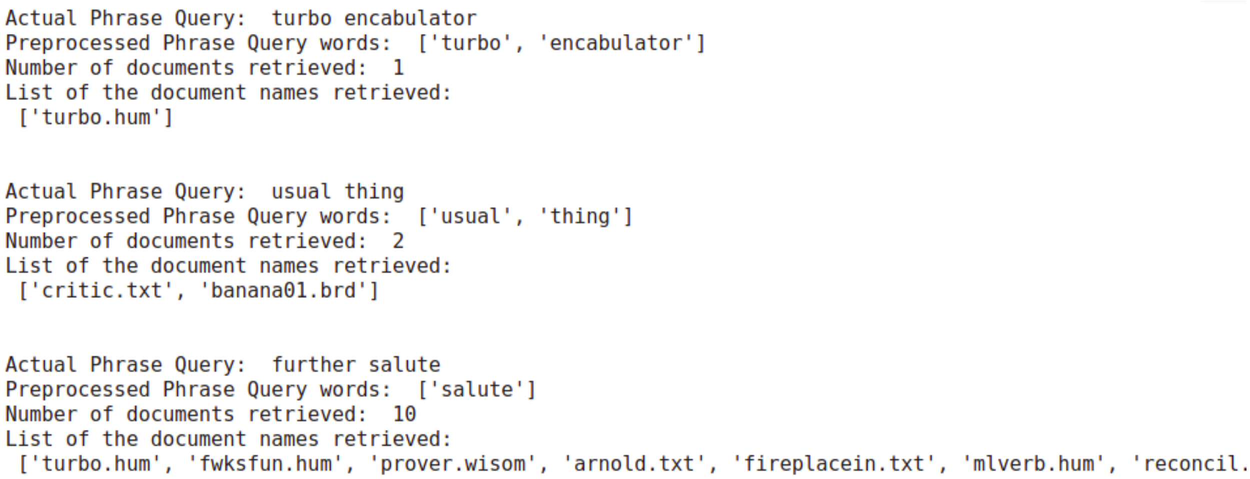
```python
      if(word_index in index_list):
        retrievingDocuments(query_index+1,query_words,document_id,word_index+1,
                            positionalIndex,retrieved_doc_ids)
```

The overall code for taking input phrase queries, preprocessing the phrase queries and giving output is :



```
phrase_queries = ['turbo encabulator', 'usual thing', 'further salute']

for query in phrase_queries:
  query_words=preprocessingQuery(query)

  pos=0
  retrieved_doc_ids = set()
  retrievingDocuments(0,query_words,-1,-1,positionalIndex,retrieved_doc_ids)

  retrieved_doc_names=[]
  for doc_id in sorted(retrieved_doc_ids):
    retrieved_doc_names.append(alldirs[doc_id].split('/')[-1])

  print("Actual Phrase Query: ",query)
  print("Preprocessed Phrase Query words: ",query_words)
  print("Number of documents retrieved: ",len(retrieved_doc_names))
  print("List of the document names retrieved: \n",retrieved_doc_names)
  print("\n")
```

The output for the above queries are:



```
Actual Phrase Query:  turbo encabulator
Preprocessed Phrase Query words:  ['turbo', 'encabulator']
Number of documents retrieved:  1
List of the document names retrieved: 
 ['turbo.hum']


Actual Phrase Query:  usual thing
Preprocessed Phrase Query words:  ['usual', 'thing']
Number of documents retrieved:  2
List of the document names retrieved: 
 ['critic.txt', 'banana01.brd']


Actual Phrase Query:  further salute
Preprocessed Phrase Query words:  ['salute']
Number of documents retrieved:  10
List of the document names retrieved: 
 ['turbo.hum', 'fwksfun.hum', 'prover.wisom', 'arnold.txt', 'fireplacein.txt', 'mlverb.hum', 'reconcil.
```