IR Assignment-1

1st chunk:

```
import zipfile
import os

# Upload the zip file to Google Colab
from google.colab import files
uploaded = files.upload()

# Extract the contents of the zip file
zip_file_name = list(uploaded.keys())[0]
with zipfile.ZipFile(zip_file_name, 'r') as zip_ref:
    zip_ref.extractall()

# Access all text files
text_files = []
for root, dirs, files in os.walk('.'):
    for file in files:
        if file.endswith('.txt'):
            text_files.append(os.path.join(root, file))

# Print the list of text files
# for file_path in text_files:
#     print(file_path)
```

```
Choose Files text_files-2...456Z-001.zip
• text_files-20240214T092456Z-001.zip(application/x-zip-compressed) - 534066 bytes, last modified: 14/2/2024 - 100% done
Saving text_files-20240214T092456Z-001.zip to text_files-20240214T092456Z-001.zip
```

Explanation: This code takes zip file from my computer and upload in the environment of google colab. Zip file contains all text files of texts as of classroom.

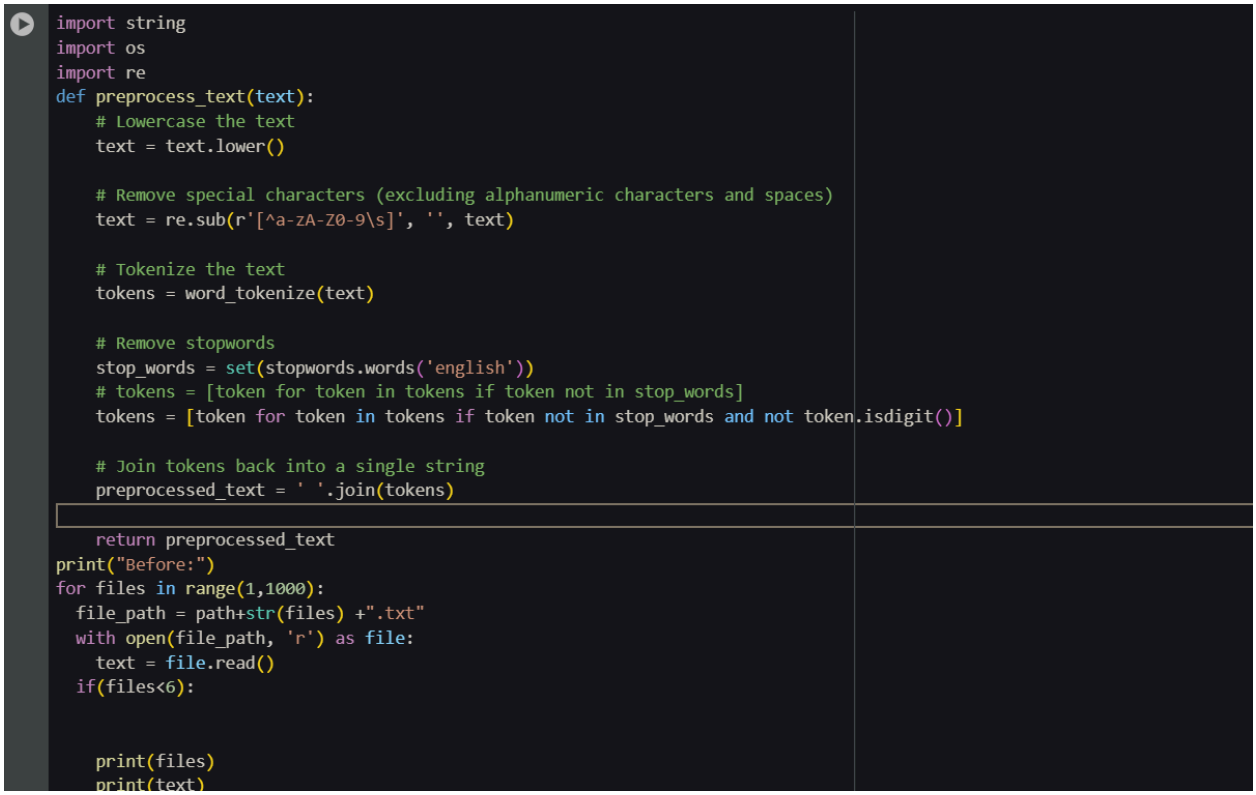
2nd Chunk:

```
path = "./text_files/file"

import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import string
import os

# Install NLTK and download required resources
nltk.download('punkt')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

Explanation: This code just take use of nltk library for preprocessing steps.

3rd Chunk:



```
import string
import os
import re
def preprocess_text(text):
    # Lowercase the text
    text = text.lower()

    # Remove special characters (excluding alphanumeric characters and spaces)
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text)

    # Tokenize the text
    tokens = word_tokenize(text)

    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    # tokens = [token for token in tokens if token not in stop_words]
    tokens = [token for token in tokens if token not in stop_words and not token.isdigit()]

    # Join tokens back into a single string
    preprocessed_text = ' '.join(tokens)

    return preprocessed_text
print("Before:")
for files in range(1,1000):
  file_path = path+str(files) +".txt"
  with open(file_path, 'r') as file:
    text = file.read()
  if(files<6):


    print(files)
    print(text)
```

Explanation: This contains the preprocessing step which uses regex format to exclude alphanumeric characters, tokenize, removing stop words and removing digits to finally create a preprocessed text.

4th Chunk:

```
from collections import defaultdict

word_docs1 = defaultdict(set)
numb =0

for items in alldoc_tokens1:
  for item in items:
    word_docs1[item].add(numb)

  numb+=1

i=0
for key, value in sorted(word_docs1.items()):
  if(i==1000):
   break

  print(f"{key}: {value}")
  i+=1
```

It is used to form word_docs which will help to know which word are present in which files.

5th:

```python
import pickle
from collections import defaultdict


with open('alldoc_tokens.pkl', 'wb') as file:
    pickle.dump(alldoc_tokens1, file)

with op Loading... ocs.pkl', 'wb') as file:
    pickle.dump(word_docs1, file)

# Loading pickled data back into memory with the same names
with open('alldoc_tokens.pkl', 'rb') as file:
    alldoc_tokens = pickle.load(file)

with open('word_docs.pkl', 'rb') as file:
    word_docs = pickle.load(file)
```

Simple pickle module to load and unload the indexes

6th:

```python
def or_set(s1, s2):
    final = set()
    l1 = sorted(s1)
    l2 = sorted(s2)

    i = 0
    j = 0
    while i < len(l1) and j < len(l2):
        if l1[i] == l2[j]:
            final.add(l1[i])
            i += 1
            j += 1
        elif l1[i] < l2[j]:
            final.add(l1[i])
            i += 1
        else:
            final.add(l2[j])
            j += 1

    while i < len(l1):
        final.add(l1[i])
        i += 1
    while j < len(l2):
        final.add(l2[j])
        j += 1

    return final

def and_set(s1, s2):
    final = set()
    l1 = sorted(s1)
```

Or operation code which just union the two sets. Similarly for and which takes intersection.
And or not, taking or with second ones complement to 0,998

And and not, taking and with second ones complement to 0,998

```python
def go_for_query(s1, s2, operation):
    if operation == "OR":
        return or_set(s1, s2)
    elif operation == "AND":
        return and_set(s1, s2)
    elif operation == "OR NOT":
        return or_not(s1, s2)
    elif operation == "AND NOT":
        return and_not(s1, s2)

def set_to_file_string(file_set):
    file_string = ""
    for num in (file_set):
        file_string += f"file{num+1}.txt,"
    # Remove the trailing comma
    file_string = file_string.rstrip(',')
    return file_string

def retrieve_documents(tokens, operations, files_set):
    # documents_retrieved = list(files_set)  # Start with all documents
    results = []  # List to store the results
    s = tokens[0]
    for i in range(1,len(tokens)):
      s = s + " "+operations[i-1]+" "+ tokens[i]

    # s= s+"\n"

    results.append(s)
```

Input taking helper code

7th:

Q3.

```python
from collections import defaultdict
def positional():
  pos_ind = dict()
  for i in range(len(alldoc_tokens)):
    lis =[]
    for j in range(len(alldoc_tokens[i])):
      if alldoc_tokens[i][j] not in pos_ind:

        pos_ind[alldoc_tokens[i][j]] ={}
        pos_ind[alldoc_tokens[i][j]][i] =[]
        pos_ind[alldoc_tokens[i][j]][i].append(j)
      else:
        if i not in pos_ind[alldoc_tokens[i][j]]:
          pos_ind[alldoc_tokens[i][j]][i] =[]
        pos_ind[alldoc_tokens[i][j]][i].append(j)

  return pos_ind

pos_ind1 = positional()
print(pos_ind1)
```

```
{'loving': {0: [0], 253: [16], 390: [2], 722: [6]}, 'vintage': {0: [1, 3], 50: [27], 149: [10], 196:
```

Creating positional indices with the help of files tokens etc

It first make dictionary when any word come up a dictionary is also made for files, when new file is added then it is initialized with a list to store indices regarding that file regarding that word.

8th Chunk:

```
def get_files(inp):
  s = preprocess_text(inp)
  tokens = word_tokenize(s)

  l=set()

  files = pos_ind[tokens[0]]
  for numb in files:
    # file_name = numb
    lis = files[numb]
    # g = pos_ind[tokens[i]]
    for ind in lis:

      for i in range(ind,ind+len(tokens)):
        if(ind>=len(alldoc_tokens[numb])):
          break
        if alldoc_tokens[numb][i]!=tokens[i-ind]:

          break

      if i==ind+len(tokens)-1:
        l.add(numb)
    # i+=1
  return l
```

A code that will found out the files regarding the query using positional index we made in 7th chunk.

9th Chunk:

```
# print(get_files(inp))

num_queries = int(input().strip())
lis=[]
for i in range(num_queries):
  s1 = input().strip()
  s2 = get_files(s1)
  lis.append(s2)

i=1
for items in lis:
  print(f"Number of documents retrieved for query {i} using positional index: ", len(items))
  print(f"Names of documents retrieved for query {i} using positional index: ", sep="")
  for item in items:
    print("file"+str(item+1)+".txt, ", sep="")

  print()

  i+=1
```

Input taking for 3rd question.