IR Assignment-1

Vishnumolakala Nikhila (MT21103)
Madadi Vineeth Reddy (MT21046)

Q1)

- ➢ Firstly, import all the necessary libraries such as nltk, pandas, numpy, etc…
- ➢ Import tokenizer, stop words, stemmer and lemmatizer from the nltk library.

Pre-processing steps:

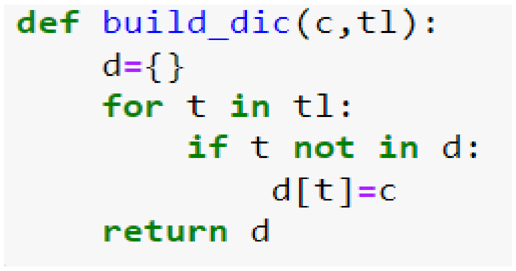
- ➢ Converted all the words to lowercase.
- ➢ Removed the punctuations and replaced it with space and also removed extra white spaces.
- ➢ Stop words are being removed.
- ➢ Word is tokenized with the help of tokenizer obtained from nltk library.
- ➢ Lemmatization is done with the help of lemmatizer obtained from the nltk library.
- ➢ Here we are mapping the document names with document numbers.

```
def pre_process(s):
    s = s.lower()
    s = s.translate(s.maketrans(string.punctuation,' '*l,''))
    s = re.sub('[^A-Za-z\s\n ]+', ' ',s)

    t = word_tokenize(s)
    t = [lem.lemmatize(w) for w in t if w not in stopwords.words('english') and w.isalpha()]
    return t
```

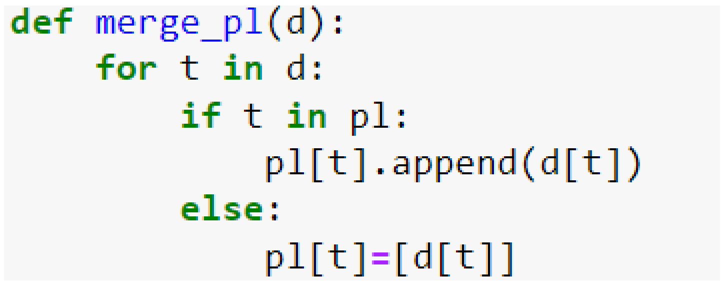
Building the Inverted index:

- ➢ First, we create a dictionary with words, we go through each word in the pre-processed words and if that word is not in the dictionary before, we add it to the dictionary.



```python
def build_dic(c,tl):
    d={}
    for t in tl:
        if t not in d:
            d[t]=c
    return d
```

➢ Then we get the posting list for each term in the dictionary by appending the document ids that the particular term is present in.

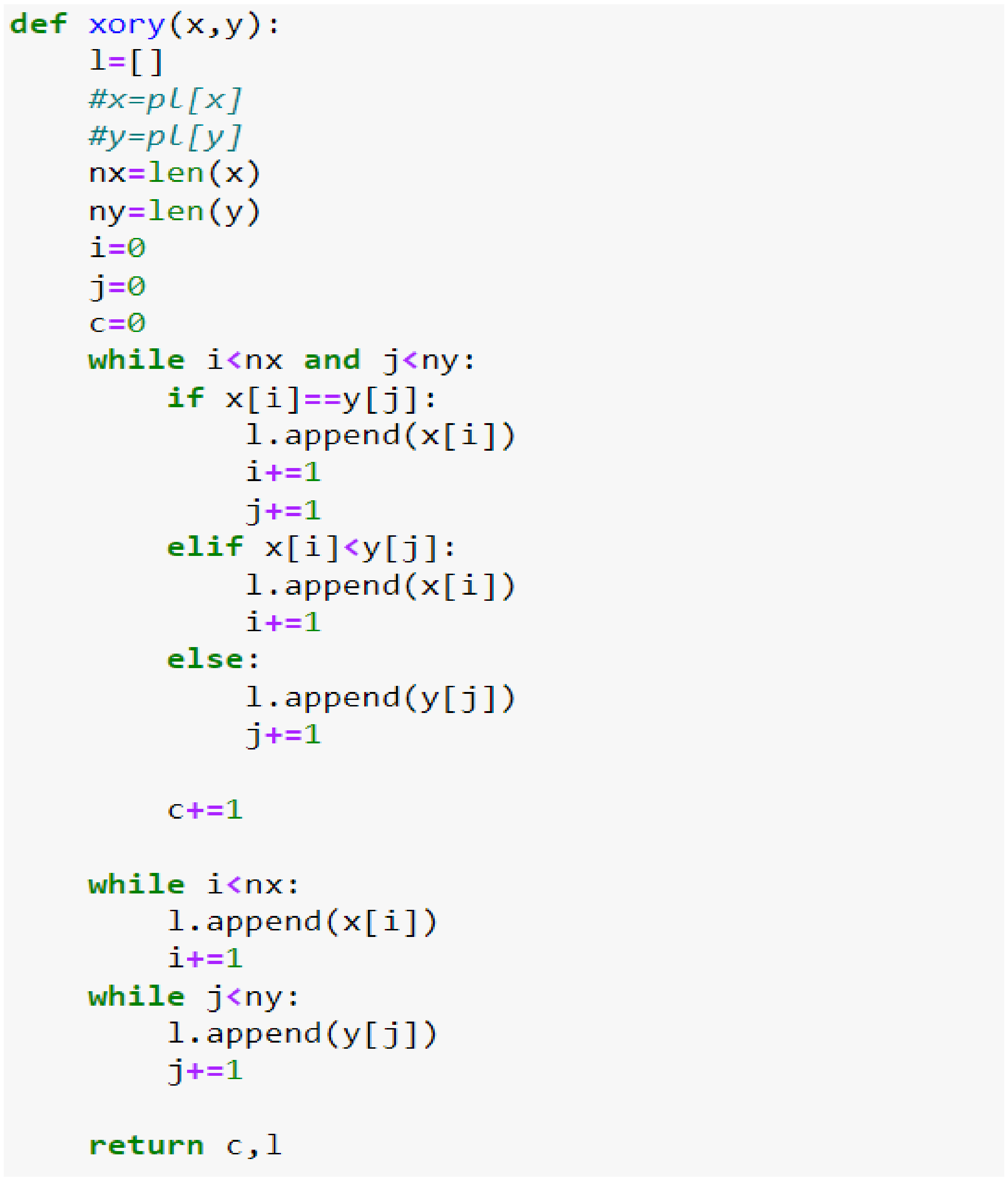


```python
def merge_pl(d):
    for t in d:
        if t in pl:
            pl[t].append(d[t])
        else:
            pl[t]=[d[t]]
```

<u>Support for the following queries:</u>

<u>i)x OR y</u>

➢ Initially we pass posting lists as parameters to the function we defined.
➢ Then we have two pointers i for x and j for y and we check if document id at those indices are equal, if equal then increment both i and j.
➢ If document id at index i is less than j, then increment i else increment j.
➢ Since this or is like union, all the leftover document ids are added at the end of the resulting list.
➢ We keep track of the number of comparisons being made and increment the counter accordingly.



```python
def xory(x,y):
    l=[]
    #x=pl[x]
    #y=pl[y]
    nx=len(x)
    ny=len(y)
    i=0
    j=0
    c=0
    while i<nx and j<ny:
        if x[i]==y[j]:
            l.append(x[i])
            i+=1
            j+=1
        elif x[i]<y[j]:
            l.append(x[i])
            i+=1
        else:
            l.append(y[j])
            j+=1

        c+=1

    while i<nx:
        l.append(x[i])
        i+=1
    while j<ny:
        l.append(y[j])
        j+=1

    return c,l
```

<u>ii)x AND y</u>

- ➢ This is similar to that of x OR y, the difference being since AND is like intersection, we don't append the remaining document ids at the end.
- ➢ Here also we keep a counter to keep track of the number of comparisons that are being made.

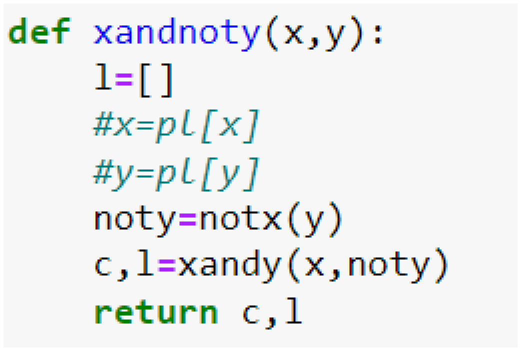
```
def xandy(x,y):
    l=[]
    #x=pl[x]
    #y=pl[y]
    nx=len(x)
    ny=len(y)
    i=0
    j=0
    c=0
    while i<nx and j<ny:
        if x[i]==y[j]:
            l.append(x[i])
            i+=1
            j+=1
        elif x[i]<y[j]:
            i+=1
        else:
            j+=1
        c+=1
    return c,l
```

iii)x AND NOT y

- First, not operation is applied to y. We take every term that is not in y.

```
def notx(x):
    l=[]
    for d in range(1,1134):
        if d not in x:
            l.append(d)
    return l
```

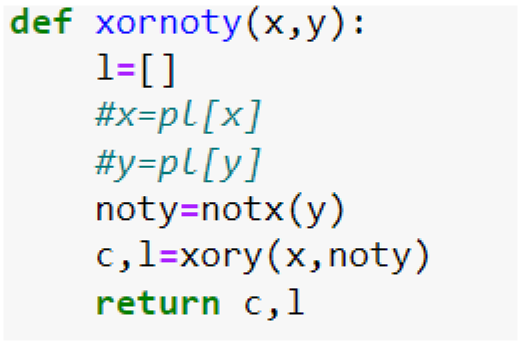
- Then we apply x AND y as usual once the above step is completed.
- We maintain a counter to keep track of number of comparisons made and then return that counter and resultant posting list.



```python
def xandnoty(x,y):
    l=[]
    #x=pl[x]
    #y=pl[y]
    noty=notx(y)
    c,l=xandy(x,noty)
    return c,l
```

iv)x OR NOT y

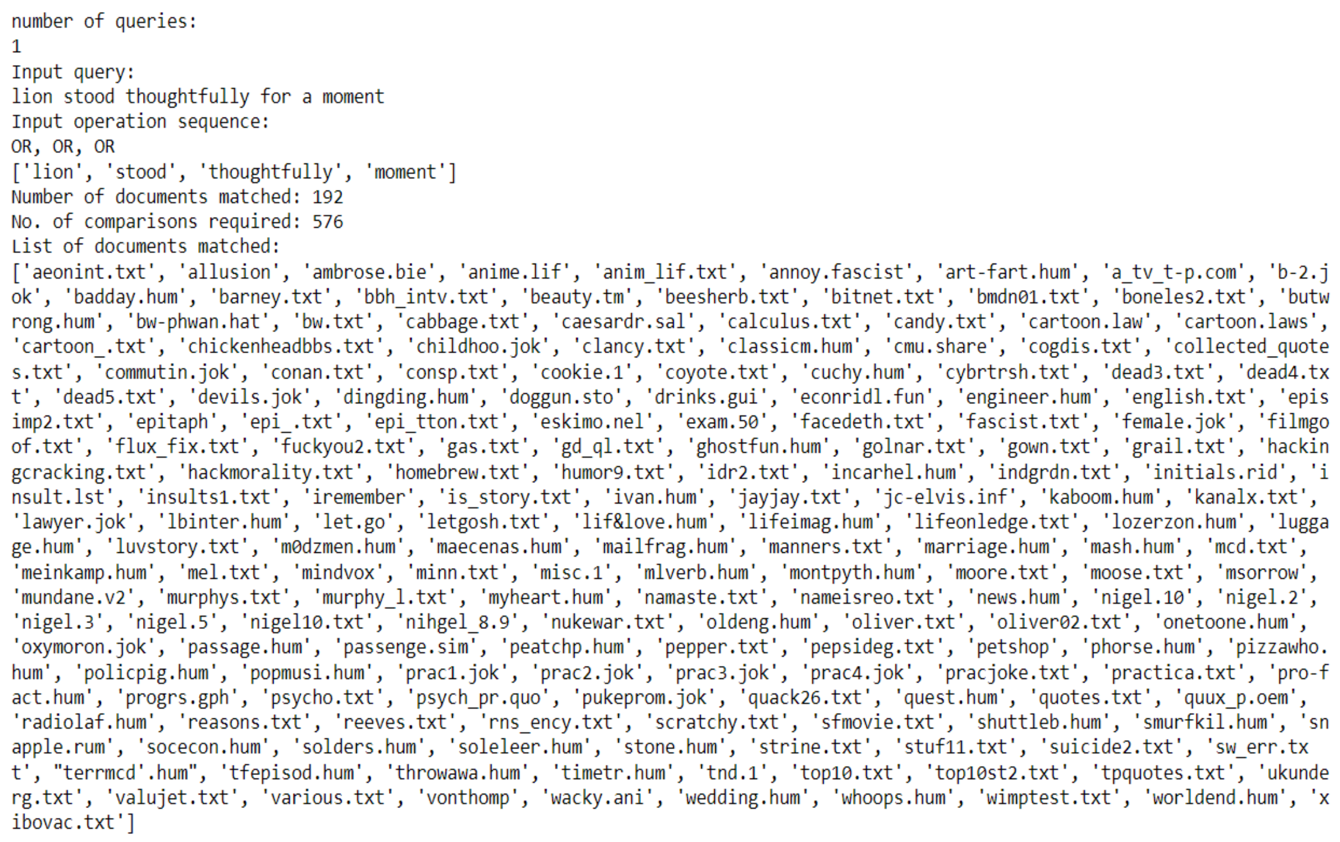
- ➢ Similar to x AND NOT y, we first get not of y and then call normally x or y.
- ➢ We return the number of comparisons made while merging along with the resultant posting list.



```python
def xornoty(x,y):
    l=[]
    #x=pl[x]
    #y=pl[y]
    noty=notx(y)
    c,l=xory(x,noty)
    return c,l
```

Sample Execution:

- ➢ We take the number of queries input from the user and then take the actual query one after another.
- ➢ We assume that the user gives proper queries with valid operations only.
- ➢ After entering the query, we give the operation sequence separated by comma between each operation.
- ➢ We apply pre-processing to the inputted query, apply a given sequence of operations, run the algorithm and finally print the number of documents matched, number of comparisons made and the list of the matched documents.



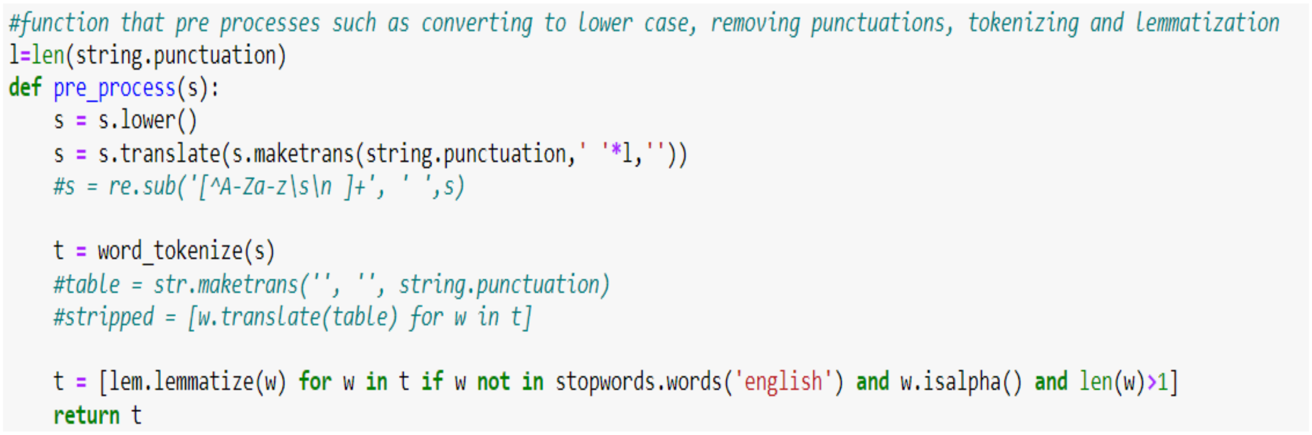
```
number of queries:
1
Input query:
lion stood thoughtfully for a moment
Input operation sequence:
OR, OR, OR
['lion', 'stood', 'thoughtfully', 'moment']
Number of documents matched: 192
No. of comparisons required: 576
List of documents matched:
['aeonint.txt', 'allusion', 'ambrose.bie', 'anime.lif', 'anim_lif.txt', 'annoy.fascist', 'art-fart.hum', 'a_tv_t-p.com', 'b-2.j
ok', 'badday.hum', 'barney.txt', 'bbh_intv.txt', 'beauty.tm', 'beesherb.txt', 'bitnet.txt', 'bmdn01.txt', 'boneles2.txt', 'butw
rong.hum', 'bw-phwan.hat', 'bw.txt', 'cabbage.txt', 'caesardr.sal', 'calculus.txt', 'candy.txt', 'cartoon.law', 'cartoon.laws',
'cartoon_.txt', 'chickenheadbbs.txt', 'childhoo.jok', 'clancy.txt', 'classicm.hum', 'cmu.share', 'cogdis.txt', 'collected_quote
s.txt', 'commutin.jok', 'conan.txt', 'consp.txt', 'cookie.1', 'coyote.txt', 'cuchy.hum', 'cybrtrsh.txt', 'dead3.txt', 'dead4.tx
t', 'dead5.txt', 'devils.jok', 'dingding.hum', 'doggun.sto', 'drinks.gui', 'econridl.fun', 'engineer.hum', 'english.txt', 'epis
imp2.txt', 'epitaph', 'epi_.txt', 'epi_tton.txt', 'eskimo.nel', 'exam.50', 'facedeth.txt', 'fascist.txt', 'female.jok', 'filmgo
of.txt', 'flux_fix.txt', 'fuckyou2.txt', 'gas.txt', 'gd_ql.txt', 'ghostfun.hum', 'golnar.txt', 'gown.txt', 'grail.txt', 'hackin
gcracking.txt', 'hackmorality.txt', 'homebrew.txt', 'humor9.txt', 'idr2.txt', 'incarhel.hum', 'indgrdn.txt', 'initials.rid', 'i
nsult.lst', 'insults1.txt', 'iremember', 'is_story.txt', 'ivan.hum', 'jayjay.txt', 'jc-elvis.inf', 'kaboom.hum', 'kanalx.txt',
'lawyer.jok', 'lbinter.hum', 'let.go', 'letgosh.txt', 'lif&love.hum', 'lifeimag.hum', 'lifeonledge.txt', 'lozerzon.hum', 'lugga
ge.hum', 'luvstory.txt', 'm0dzmen.hum', 'maecenas.hum', 'mailfrag.hum', 'manners.txt', 'marriage.hum', 'mash.hum', 'mcd.txt',
'meinkamp.hum', 'mel.txt', 'mindvox', 'minn.txt', 'misc.1', 'mlverb.hum', 'montpyth.hum', 'moore.txt', 'moose.txt', 'msorrow',
'mundane.v2', 'murphys.txt', 'murphy_l.txt', 'myheart.hum', 'namaste.txt', 'nameisreo.txt', 'news.hum', 'nigel.10', 'nigel.2',
'nigel.3', 'nigel.5', 'nigel10.txt', 'nihgel_8.9', 'nukewar.txt', 'oldeng.hum', 'oliver.txt', 'oliver02.txt', 'onetoone.hum',
'oxymoron.jok', 'passage.hum', 'passenge.sim', 'peatchp.hum', 'pepper.txt', 'pepsideg.txt', 'petshop', 'phorse.hum', 'pizzawho.
hum', 'policpig.hum', 'popmusi.hum', 'prac1.jok', 'prac2.jok', 'prac3.jok', 'prac4.jok', 'pracjoke.txt', 'practica.txt', 'pro-f
act.hum', 'progrs.gph', 'psycho.txt', 'psych_pr.quo', 'pukeprom.jok', 'quack26.txt', 'quest.hum', 'quotes.txt', 'quux_p.oem',
'radiolaf.hum', 'reasons.txt', 'reeves.txt', 'rns_ency.txt', 'scratchy.txt', 'sfmovie.txt', 'shuttleb.hum', 'smurfkil.hum', 'sn
apple.rum', 'socecon.hum', 'solders.hum', 'soleleer.hum', 'stone.hum', 'strine.txt', 'stuf11.txt', 'suicide2.txt', 'sw_err.tx
t', "terrmcd'.hum", 'tfepisod.hum', 'throwawa.hum', 'timetr.hum', 'tnd.1', 'top10.txt', 'top10st2.txt', 'tpquotes.txt', 'ukunde
rg.txt', 'valujet.txt', 'various.txt', 'vonthomp', 'wacky.ani', 'wedding.hum', 'whoops.hum', 'wimptest.txt', 'worldend.hum', 'x
ibovac.txt']
```

Q2)

- ➢ Firstly, all the necessary libraries such as nltk, pandas, numpy, etc…are imported.
- ➢ Import tokenizer, stop words, stemmer and lemmatizer from the nltk library.

<u>Pre-processing steps:</u>

- ➢ Initially, all the words are converted to lowercase.
- ➢ Word_tokenize imported from nltk is used to perform tokenization on the given dataset.
- ➢ Next, the stop words present are removed from the dataset
- ➢ Removed the punctuations present and they are replaced with space and also extra white spaces are removed.
- ➢ Blank space tokens are being removed by checking if length greater than one or not.
- ➢ Here we are mapping the document names with document numbers.



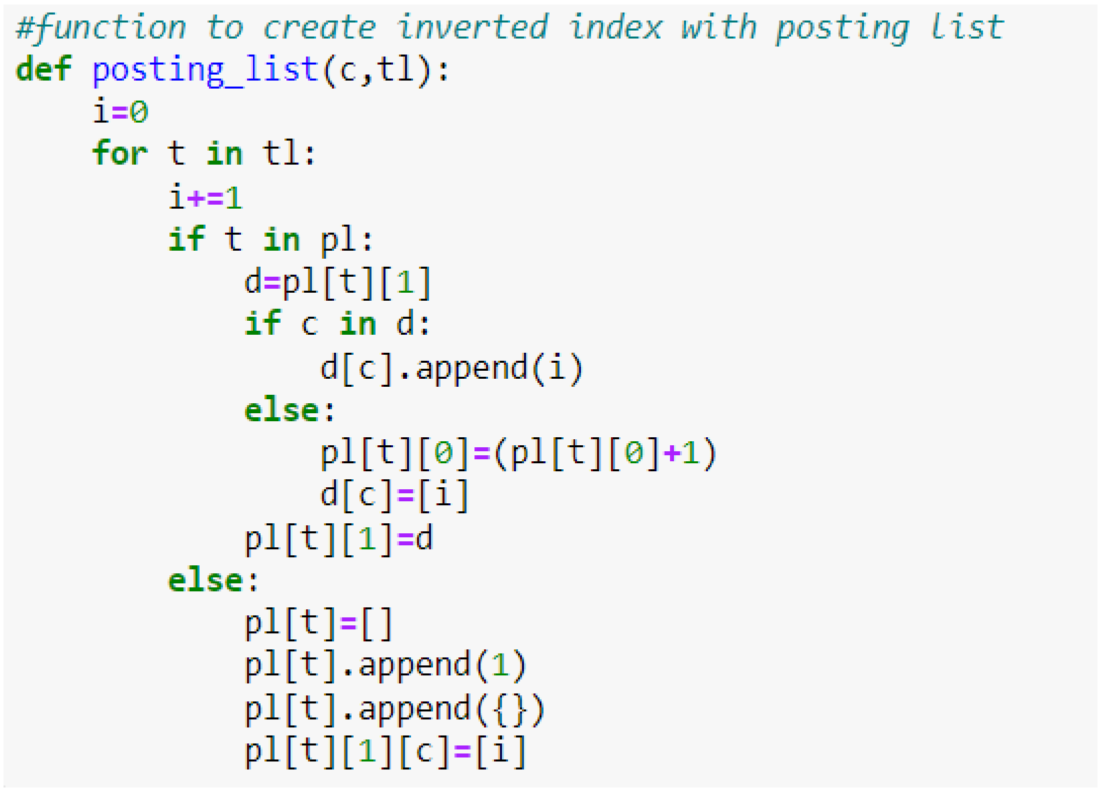
```python
#function that pre processes such as converting to lower case, removing punctuations, tokenizing and lemmatization
l=len(string.punctuation)
def pre_process(s):
    s = s.lower()
    s = s.translate(s.maketrans(string.punctuation,' '*l,''))
    #s = re.sub('[^A-Za-z\s\n ]+', ' ',s)

    t = word_tokenize(s)
    #table = str.maketrans('', '', string.punctuation)
    #stripped = [w.translate(table) for w in t]

    t = [lem.lemmatize(w) for w in t if w not in stopwords.words('english') and w.isalpha() and len(w)>1]
    return t
```

Building the Positional index:

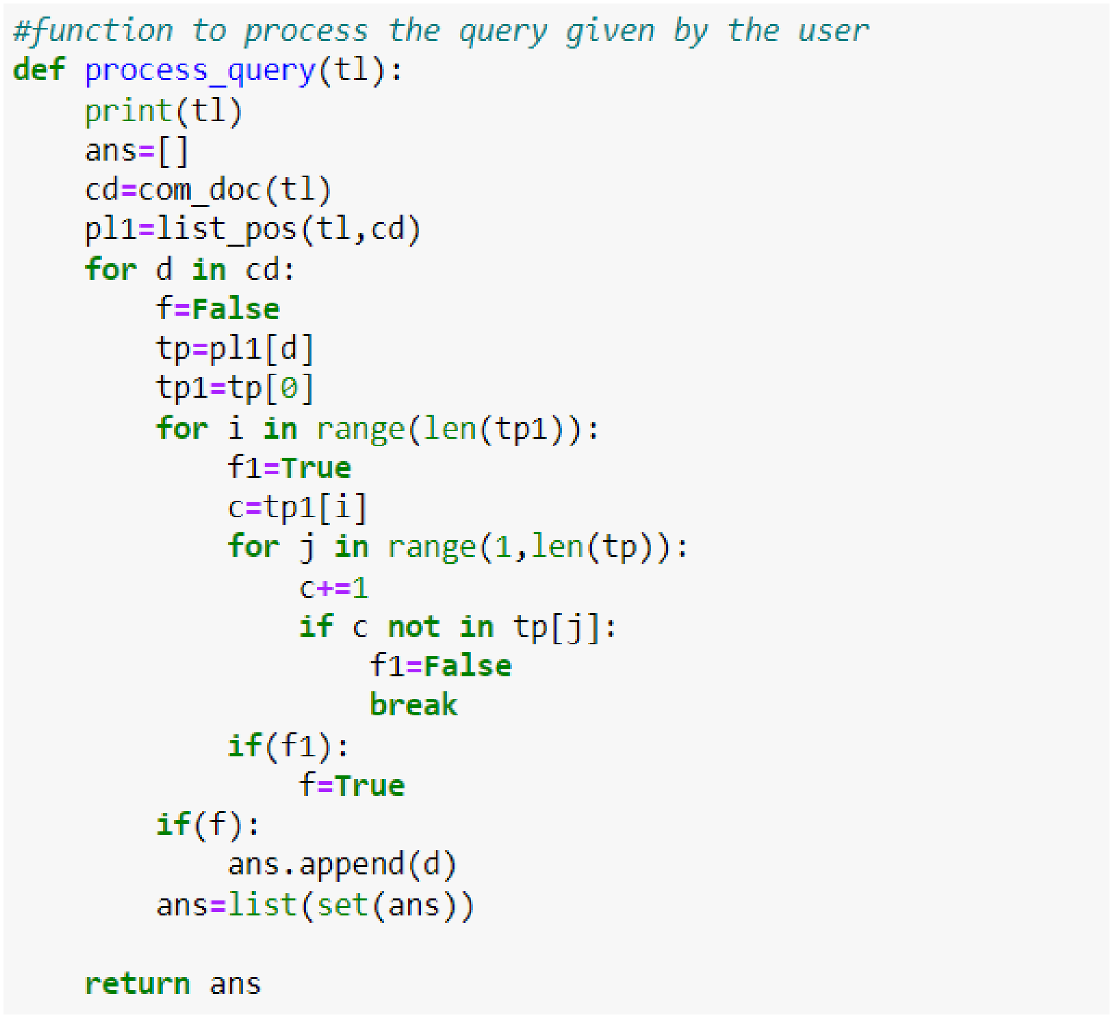
- ➢ Collected all the pre-processed tokens from each document.
- ➢ For each token, it is stored in the dictionary.
- ➢ It further contains a dictionary of documents in which the token is present and it's position in the document.



```python
#function to create inverted index with posting list
def posting_list(c,tl):
    i=0
    for t in tl:
        i+=1
        if t in pl:
            d=pl[t][1]
            if c in d:
                d[c].append(i)
            else:
                pl[t][0]=(pl[t][0]+1)
                d[c]=[i]
            pl[t][1]=d
        else:
            pl[t]=[]
            pl[t].append(1)
            pl[t].append({})
            pl[t][1][c]=[i]
```

Processing the query:

- ➢ First we collected all the common documents in which all the given tokens are present.
- ➢ Then, we took the positional indexes token wise in a list for each document.
- ➢ Finally, we checked for the consecutive positions and stored the corresponding document in the result, If all the tokens are present consecutively.



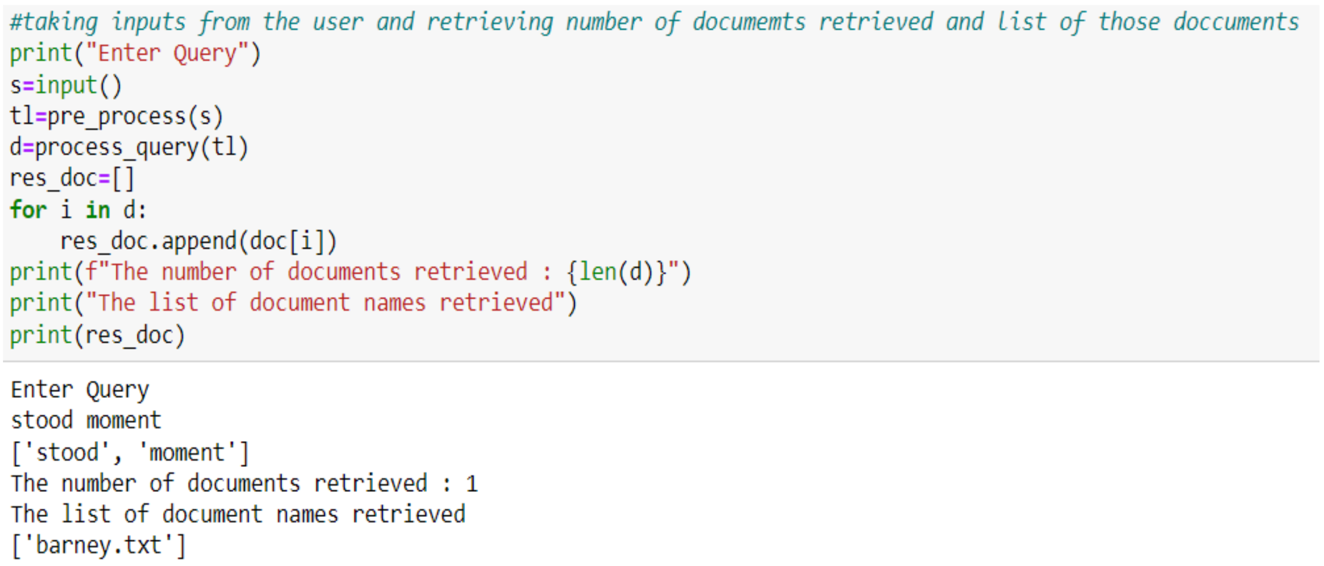
```
#function to process the query given by the user
def process_query(tl):
    print(tl)
    ans=[]
    cd=com_doc(tl)
    pl1=list_pos(tl,cd)
    for d in cd:
        f=False
        tp=pl1[d]
        tp1=tp[0]
        for i in range(len(tp1)):
            f1=True
            c=tp1[i]
            for j in range(1,len(tp)):
                c+=1
                if c not in tp[j]:
                    f1=False
                    break
            if(f1):
                f=True
        if(f):
            ans.append(d)
        ans=list(set(ans))

    return ans
```

Sample Execution:

- ➢ First, we take the query from the user, then process the query and finally return the number of documents retrieved and list of documents that are retrieved as the result.

- We assume that the user gives proper queries with valid operations only.



```python
#taking inputs from the user and retrieving number of documemts retrieved and list of those doccuments
print("Enter Query")
s=input()
tl=pre_process(s)
d=process_query(tl)
res_doc=[]
for i in d:
    res_doc.append(doc[i])
print(f"The number of documents retrieved : {len(d)}")
print("The list of document names retrieved")
print(res_doc)
```

```
Enter Query
stood moment
['stood', 'moment']
The number of documents retrieved : 1
The list of document names retrieved
['barney.txt']
```