Term Document Incidence Matrix and query evaluation

```
import numpy as np
import pandas as pd
def check_parenthesis(s):
    l=[]
    for i in s:
        if i=='(':
            l.append(i)
        elif i==')':
            l.pop()
        else:
            continue
    if len(l)==0:
        return True
    else:
        return False

def Precedence(a):
    if a.lower()=='and': return 1
    elif a.lower()=='or': return 2
    elif a.lower()=='not': return 3
    else:return 0

def Operator(a):
    if a.lower() in ['(',')','and','or','not']: return True
    else: return False

def infix_to_postfix(s):
    operators=[]
    operands=[]
    for i in s:
        if i=='(': operators.append(i)
        elif i==')':
            while(len(operators)!=0 and operators[-1]!='('):
                operands.append(operators.pop())
            operators.pop()
        elif not Operator(i):
            operands.append(i)
        else:
            while(len(operators)!=0 and
Precedence(i)<=Precedence(operators[-1])):
                operands.append(i)
            operators.append(i)
    while(len(operators)!=0):
           operands.append(operators.pop())
```

```
    return operands


def AND_OF(x,y):
    l1=np.zeros(len(docs),dtype=int)
    l2=np.zeros(len(docs),dtype=int)
    if type(x) is str:
        for i in range(len(docs)):
            if x.lower() in docs[i].lower():
                l1[i]=1
    else:
        l1=x
    if type(y) is str:
        for i in range(len(docs)):
            if y.lower() in docs[i].lower():
                l2[i]=1
    else:
        l2=y
    l1=np.array(l1)
    l2=np.array(l2)
    return list(l1&l2)

def OR_OF(x,y):
    l1=np.zeros(len(docs),dtype=int)
    l2=np.zeros(len(docs),dtype=int)
    if type(x) is str:
        for i in range(len(docs)):
            if x.lower() in docs[i].lower():
                l1[i]=1
    else:
        l1=x
    if type(y) is str:
        for i in range(len(docs)):
            if y.lower() in docs[i].lower():
                l2[i]=1
    else:
        l2=y
    l1=np.array(l1)
    l2=np.array(l2)
    return list(l1|l2)

def NOT_OF(x):
    l1=np.zeros(len(docs),dtype=int)
    if type(x) is str:
            for i in range(len(docs)):
                if x.lower() not in docs[i].lower():
                    l1[i]=1
    else:
            for i in range(len(docs)):
                if l1[i]==1:
```

```
                    l1[i]=0
                else:
                    l1[i]=1
    return l1


def queryPrint(r):
    print('The Documents related to the given query are:')
    if 1 in r:
        for i in range(len(docs)):
            if r[i]==1:
                print(f'{docNames[i]}')
    else:
        print('No related documents found for the given query.')

def queryEval(s):
    if(check_parenthesis(s)):
        s=infix_to_postfix(s)
        operands=[]
        for i in s:
            if not Operator(i):
                operands.append(i)
            elif i.lower()=='and':
                b=operands.pop()
                res=AND_OF(operands.pop(),b)
                operands.append(res)
            elif i.lower()=='or':
                b=operands.pop()
                res=OR_OF(operands.pop(),b)
                operands.append(res)
            else:
                res=NOT_OF(operands.pop())
                operands.append(res)
        res=operands.pop()
        queryPrint(res)
    else:
        print('Invalid no of Parenthesis in the given query.')
        return

docNames=['Doc1','Doc2','Doc3','DOC4']
docs=[]
for i in docNames:
    try:
        with open(f'{i}.txt') as file:
            data=file.readlines()
            docs.extend(data)
    except:
        print(f'There is a Problem opening the file {i}.txt')

for i in range(len(docNames)):
```

```
    print(f'{docNames[i]}:\n{docs[i]}')
docTerms=[]
for i in docs:
    for j in i.lower().strip().split():
        if j not in docTerms:
            docTerms.append(j)
docTerms.sort()


TDIM=np.zeros((len(docTerms),len(docNames)),dtype=int)
for i in range(len(docTerms)):
    for j in range(len(docs)):
        if docTerms[i] in docs[j].lower():
            TDIM[i][j]=1

df=pd.DataFrame(TDIM,index=docTerms,columns=docNames)
display(df)


s=input("Enter you query:").split()
queryEval(s)
#( established and many ) or not lorem
```

```
Doc1:
It is a long established fact that a reader will be distracted by the readable content of a page have suffered when looking at its layout
Doc2:
Many desktop publishing packages and web page editors have suffered alteration now use Lorem Ipsum as their default model text, and a search for lorem ipsum will uncover many web sites still in their infancy
Doc3:
There are many variations of passages of Lorem Ipsum available but the majority have suffered alteration in some form by injected humour or randomised words which donot look even slightly believable
DOC4:
But I must explain to you how all this mistaken idea of denouncing pleasure and praising pain was born and I will give you a complete account of the system and expound the actual teachings of the great explorer of the truth the master builder of human happiness
```

	Doc1	Doc2	Doc3	DOC4
a	1	1	1	1
account	0	0	0	1
actual	0	0	0	1
all	0	0	0	1
alteration	0	1	1	0
...	...	...	...	...
when	1	0	0	0
which	0	0	1	0

```
will            1     1     0     1
words           0     0     1     0
you             1     0     0     1

[98 rows x 4 columns]

Enter you query:( established and many ) or not lorem
The Documents related to the given query are:
Doc1
DOC4
```