|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Practical No | Topic | Date | Page No | Remark |
| 1 | Write a program to demonstrate bitwise operation. | 21/12/2021 | 2-3 |  |
| 2 | Implement Page Rank Algorithm | 01/02/2022 | 4-5 |  |
| 3 | Implement Dynamic programming algorithm for computing the edit distance between strings s1 and s2. (Hint. Levenshtein Distance) | 14/12/2021 | 6 |  |
| 4 | Write a program to Compute Similarity between two text documents. | 08/02/2022 | 7-8 |  |
| 5 | Write a map-reduce program to count the number of occurrences of each alphabetic character in the given dataset. The count for each letter should be case-insensitive (i.e., include both upper-case and lower-case versions of the letter; Ignore non-alphabetic characters). | 14/12/2021 | 9 |  |
| 6 | Write a program for Pre-processing of a Text Document: stop word removal. | 23/11/2021 | 10-11 |  |
| 7 | Write a program for mining Twitter to identify tweets for a specific period and identify trends and named entities. | 11/1/2022 | 12-13 |  |
| 8 | Write a program to implement simple web crawler. | 07/12/2021 | 14 |  |
| 9 | Write a program to parse XML text, generate Web graph and compute topic specific page rank. | 15/02/2022 | 15-17 |  |

**Practical 1**

**AIM:** Write a program to demonstrate bitwise operation

**WRITE UP:**

In computer programming, a bitwise operation operates on a bit string, a bit array or a binary numeral at the level of its individual bits. It is a fast and simple action, basic to the higher-level arithmetic operations and directly supported by the processor.

**INPUT:**

import pandas as pd

from sklearn.feature\_extraction.text import CountVectorizer

docs=['why hello hello there','omg hello pony','she went there?omg']

print(docs)

vec=CountVectorizer()

x=vec.fit\_transform(docs)

print('vocabulary',vec.vocabulary\_)

print(x)

df=pd.DataFrame(x.toarray(),columns=vec.get\_feature\_names())

print(df)

w1=input("enter word1: ")

w2=input("enter word1: ")

op=input("enter operator: ")

x=[]

for i in range(df.shape[0]):

if(op=="&"):

a=(list(df.loc[:,w1]))[i]&(list(df.loc[:,w2]))[i]

x.append(a)

if(op=="|"):

a=(list(df.loc[:,w1]))[i]|(list(df.loc[:,w2]))[i]

x.append(a)

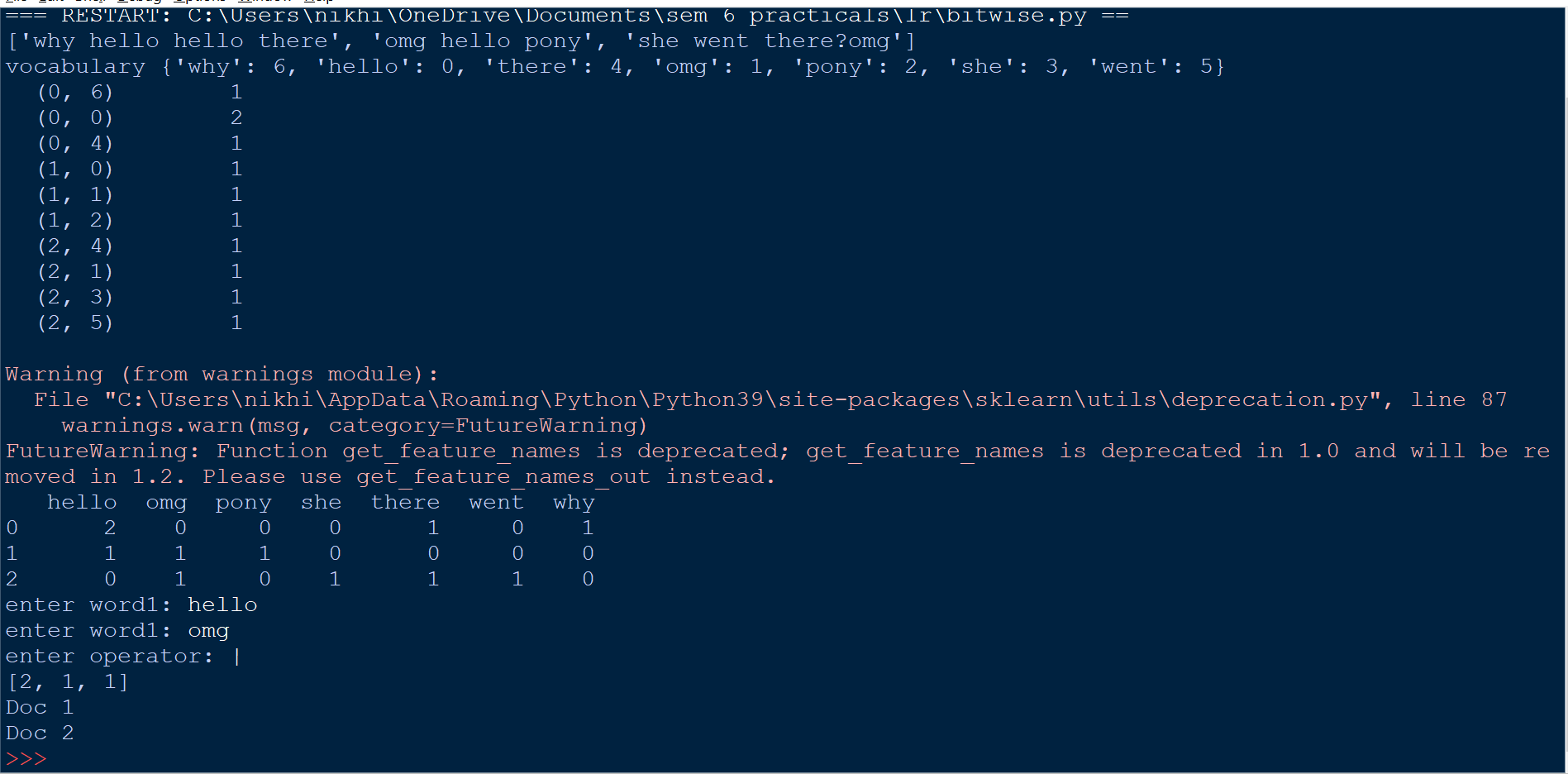
print(x)

for i in range(df.shape[0]):

if(x[i]==1):

print("Doc",i)

**OUTPUT:**

****

**Practical 2**

**AIM:** Implement Page Rank Algorithm.

**WRITE UP:**

PageRank is an algorithm used by Google Search to rank web pages in their search engine results. It is named after both the term "web page" and co-founder Larry Page. PageRank is a way of measuring the importance of website pages

**INPUT:**

import numpy as np

from fractions import Fraction

def display\_format (my\_vector, my\_decimal):

return np.round ( (my\_vector).astype (float), decimals=my\_decimal)

dp= Fraction (1,3)

M= np.matrix ([[0, Fraction (1,2), Fraction(1,2)],

[1, 0, 0],

[1, 0, 0]]

)

print (M)

E = np.zeros((3,3))

E[:] = dp

beta=0.9

A = beta \* M + ((1-beta)\* E)

r = np.matrix ([dp, dp, dp])

r = np. transpose (r)

previous\_r= r

for i in range (1,10):

r = A \* r

print (r)

print (display\_format(r, 3))

if (previous\_r==r).all():

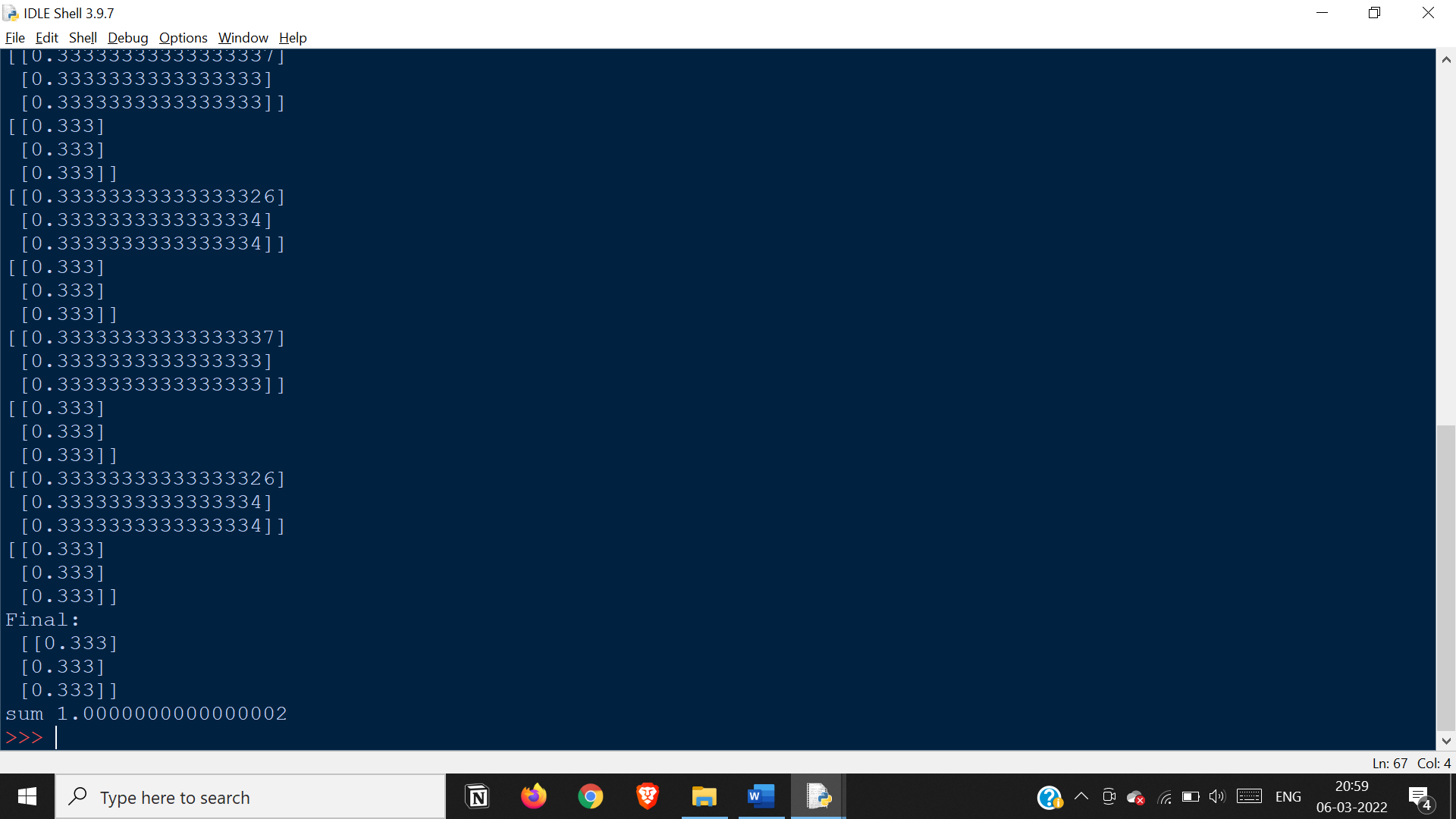
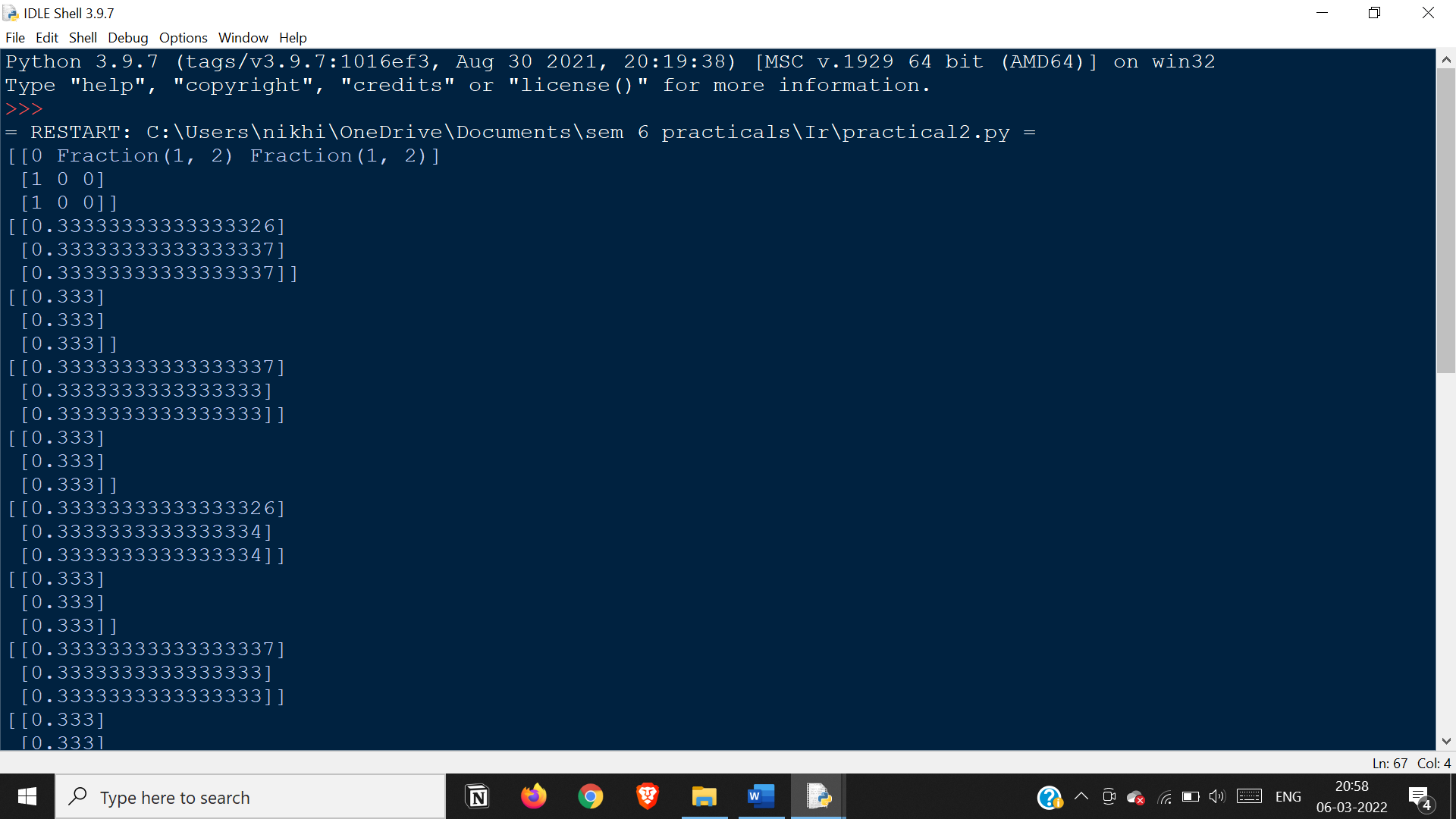
break

previous\_r = r

print ("Final: \n", display\_format(r, 3))

print ("sum", np.sum (r))

**OUTPUT:**



**Practical 3**

**AIM:** Implement Dynamic programming algorithm for computing the edit distance between

strings s1 and s2. (Hint. Levenshtein Distance)

**WRITE UP:**

In information theory, linguistics, and computer science, the Levenshtein distance is a string metric for measuring the difference between two sequences. Informally, the Levenshtein distance between two words is the minimum number of single-character edits required to change one word into the other.

**INPUT:**

def editDistance(str1,str2,m,n):

if m==0:

return 0

if n==0:

return m

if(str1[m-1]==str2[n-1]):

return editDistance(str1,str2,m-1,n-1)

return 1+ min(editDistance(str1,str2,m,n-1),#insert

editDistance(str1,str2,m-1,n),#remove

editDistance(str1,str2,m-1,n-1)) #replace

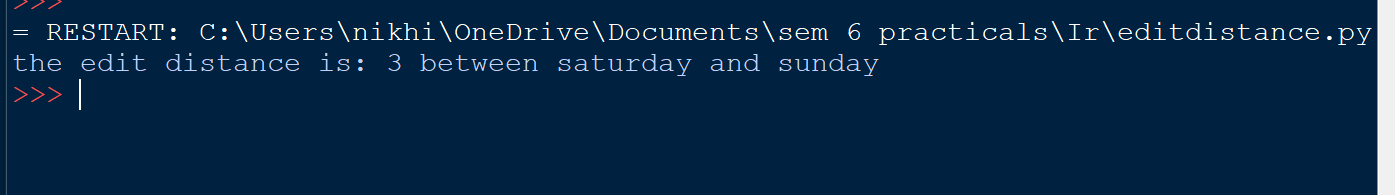
str1="saturday"

str2="sunday"

print("the edit distance is:",editDistance(str1,str2,len(str1),len(str2))

, "between",str1, "and",str2)

**OUTPUT:**



**Practical 4**

**AIM:** Write a program to Compute Similarity between two text documents

**WRITE UP:**

The traditional approach to compute text similarity between documents is to do so by transforming the input documents into real-valued vectors. The goal is to have a vector space where similar documents are “close”, according to a chosen similarity measure.

This approach takes the name of Vector Space Model, and it’s very convenient because it allows us to use simple linear algebra to compute similarities. We just have to define two things:

A way of transforming documents into vectors

A similarity measure for vectors

The simplest way to build a vector from text is to use word counts.

**File1.txt:** Information retrieval in computing and information science is the process of obtaining information system resources that are relevant to an information need from a collection of those resources.

**File2.txt**: Information retrieval in computing and information science is the process of obtaining information system resources that relevant to an information need from of those resources

**INPUT:**

import math

import string

import sys

def read\_file(filename):

try:

with open(filename,'r') as f:

data=f.read()

return data

except IOError:

print("Error opening file",filename)

sys.exit()

translation\_table=str.maketrans(string.punctuation+string.ascii\_uppercase," "\*len(string.punctuation)+string.ascii\_lowercase)

def get\_words\_from\_line\_list(text):

text=text.translate(translation\_table)

word\_list=text.split()

return word\_list

def count\_frequency(word\_list):

D={}

for new\_word in word\_list:

if new\_word in D:

D[new\_word]+D[new\_word]+1

else:

D[new\_word]=1

return D

def word\_frequency\_for\_file(filename):

line\_list=read\_file(filename)

word\_list=get\_words\_from\_line\_list(line\_list)

freq\_mapping=count\_frequency(word\_list)

print("File",filename,":",)

print(len(line\_list),"letters,",)

print(len(word\_list),"words,",)

print(len(freq\_mapping),"distinct words")

return freq\_mapping

def dotProduct(D1,D2):

sum=0.0

for key in D1:

if key in D2:

sum+=(D1[key]\*D2[key])

return sum

def vector\_angle(D1,D2):

numerator=dotProduct(D1,D2)

denominator=math.sqrt(dotProduct(D1,D1)\*dotProduct(D2,D2))

return math.acos(numerator/denominator)

def documentSimilarity(filename\_1,filename\_2):

sorted\_word\_list\_1=word\_frequency\_for\_file(filename\_1)

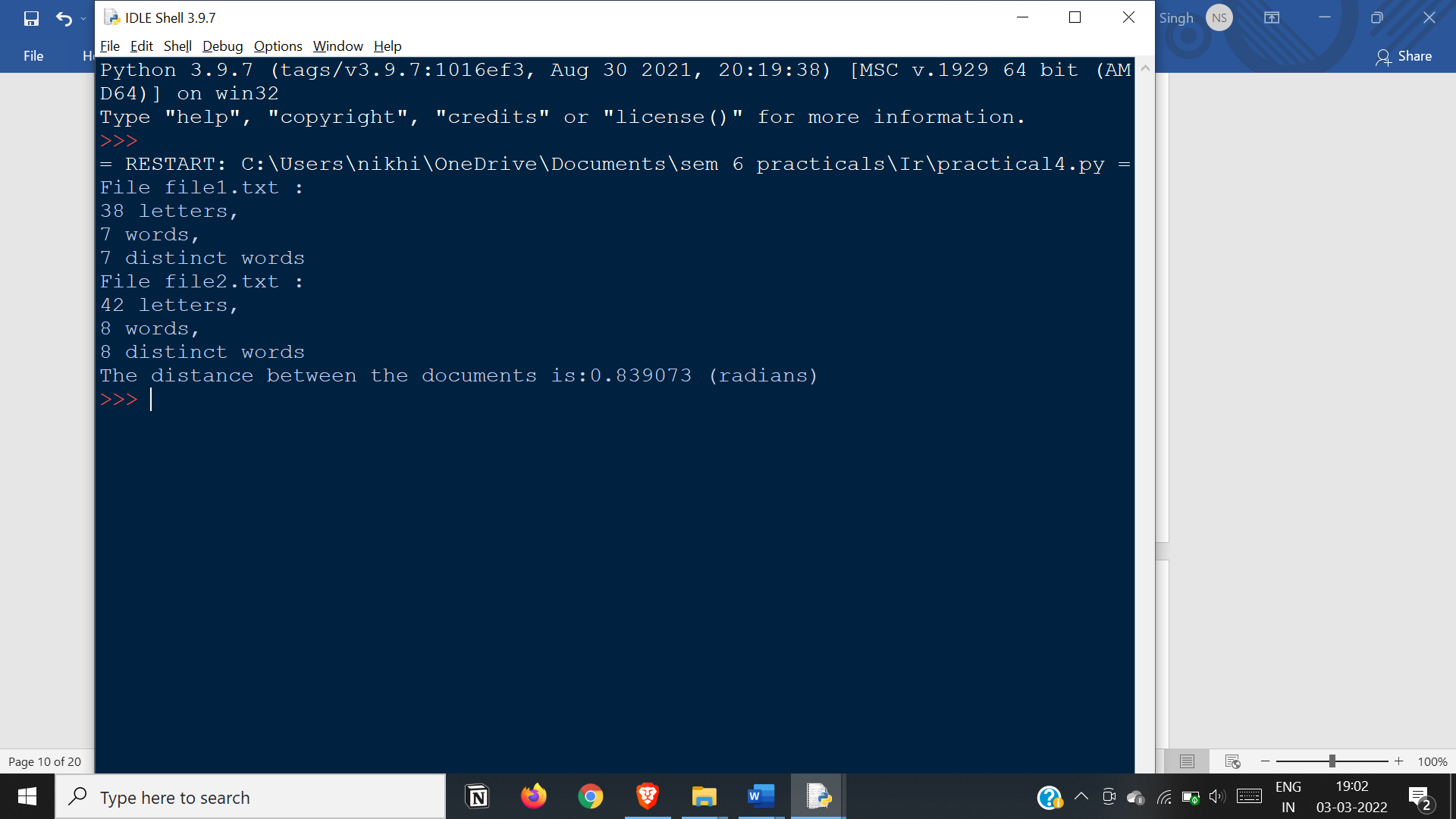
sorted\_word\_list\_2=word\_frequency\_for\_file(filename\_2)

distance=vector\_angle(sorted\_word\_list\_1,sorted\_word\_list\_2)

print("The distance between the documents is:%0.6f (radians)"%distance)

documentSimilarity("file1.txt","file2.txt")

**OUTPUT:**



**Practical 5**

**AIM:** Write a map-reduce program to count the number of occurrences of each alphabetic character in the given dataset. The count for each letter should be case-insensitive (i.e., include both upper-case and lower-case versions of the letter; Ignore non-alphabetic characters).

**WRITE UP:**

Our task is to count the frequency of each character present in our input file. We are using python for implementing this particular scenario. However, The MapReduce program can also be written in Java or C++. Execute the below steps to complete the task for finding the occurrence of each character.

**INPUT:**

from collections import Counter

import re

test = input("Enter a String: ")

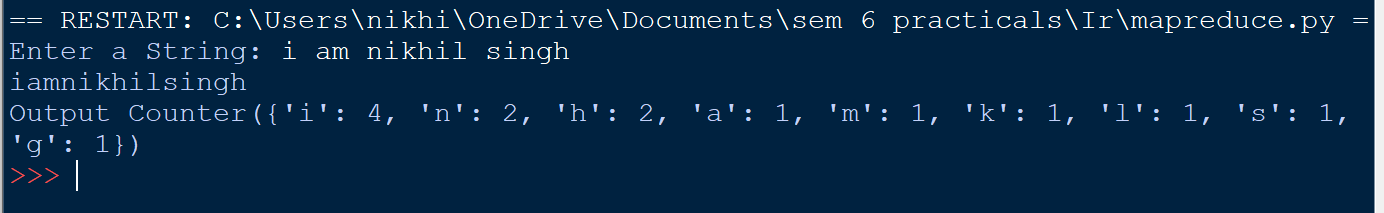
string = re.sub("[^a-zA-Z]+", "", test)

print(string)

res = Counter(string.casefold())

print ("Output " + str(res))

**OUTPUT:**



**Practical 6**

**AIM:** Write a program for Pre-processing of a Text Document: stop word removal.

**WRITE UP:**

Stop words are any word in a stop list which are filtered out before or after processing of natural language data. There is no single universal list of stop words used by all natural language processing tools, nor any agreed upon rules for identifying stop words, and indeed not all tools even use such a list.

**INPUT:**

import nltk

nltk.download ('stopwords')

nltk.download ('punkt')

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

file=open(r"C:\Users\nikhi\OneDrive\Documents\sem 6 practicals\Ir\message.txt",'r')

sent=file.read()

stop=set (stopwords.words ('english'))

token=word\_tokenize (sent)

a=[]

for w in token:

if w not in stop:

a.append (w)

print("Original sentence : ", token)

print("="\*45)

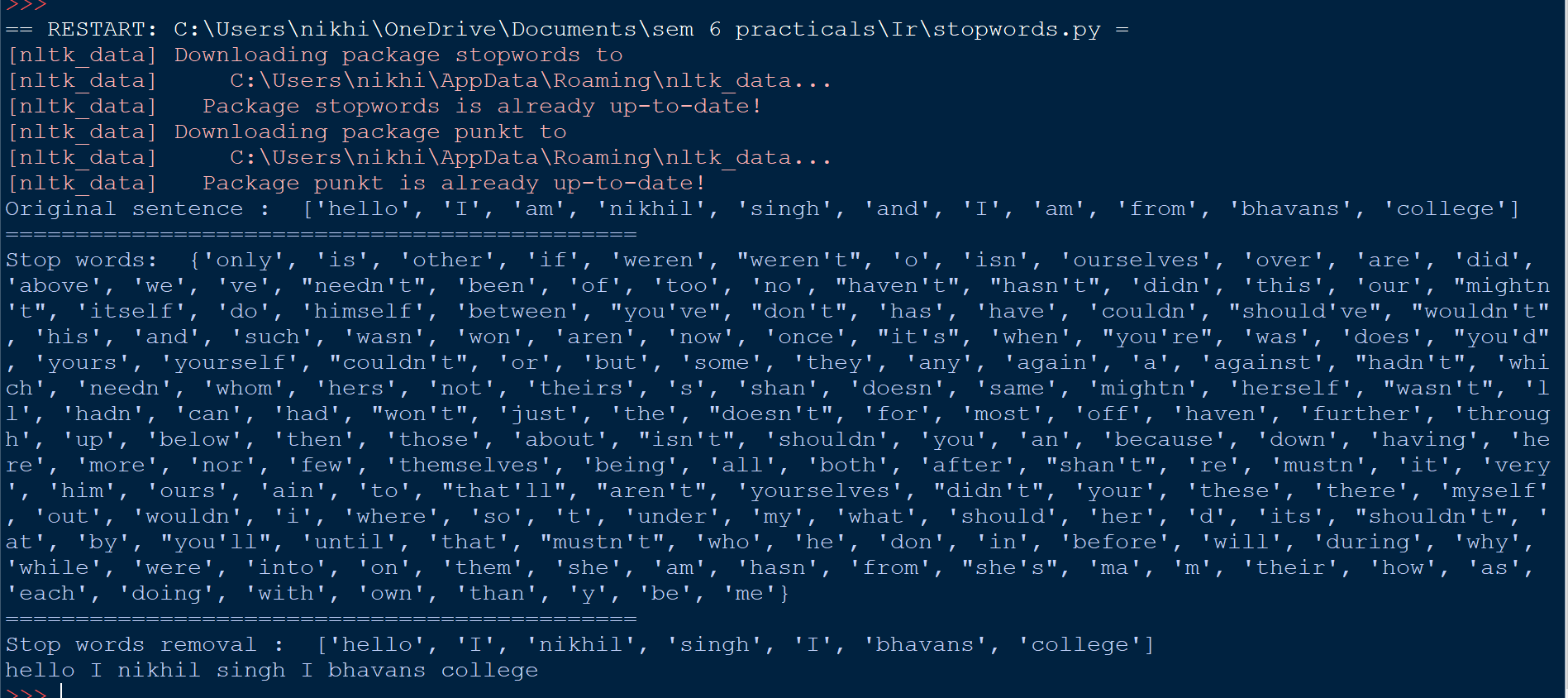
print ("Stop words: ",stop)

print("="\*45)

print ("Stop words removal : ",a)

print(" ".join(a))

**OUTPUT:**



**Practical 7**

**AIM:** Write a program for mining Twitter to identify tweets for a specific period and identify trends and named entities.

**WRITE UP:**

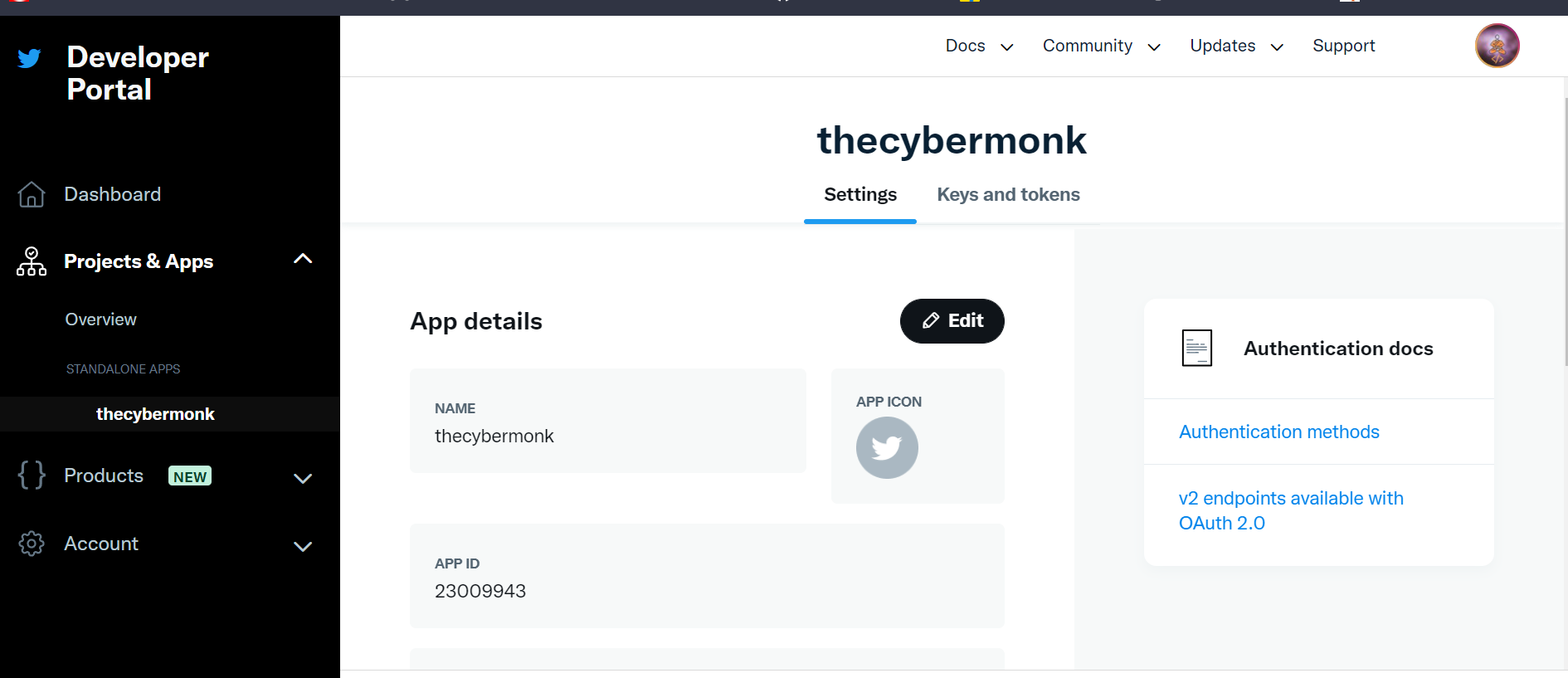
**Making a Developer account:**

A developer account is used for managing your Twitter API access

To sign up for a developer account:

1. Log-in to Twitter and verify your email address and phone number
2. Go sign up at developer.twitter.com with your basic name, location and use case details
3. Review and accept the developer agreement
4. You should now have access to the Developer Portal to create a new App and Project with Essential access

To check if you have a developer account go to the developer portal dashboard to review your account status and setup.



**INPUT:**

import tweepy

consumer\_key="vpxW7hxyer3AJJuAfo1PVIkWk"

consumer\_secret="oRgej1MsGPx8nuLmfYMFCiR2tIpXmggjW2n4Rs9r7MJwOraIiQ"

access\_token="1351248261403664384-eItGUIrp0m0iBhXrhIZPj8ghLVRwut"

access\_token\_secret= "XPkovcRJlRpIS3x6kJhiFxZytUkxH2N0mn4EFlL5Xyfm6"

client\_id="UnBUQmptSzEzd3E4c2Jlb3owZ3k6MTpjaQ"

client\_secret="uFxb2VtE0oySBgZgu2014O4scT41YaUUmLlha14KZVjUM-hb42"

#Creating the authentication object

auth=tweepy.OAuthHandler (consumer\_key, consumer\_secret)

#setting your acces token and secret

auth.set\_access\_token (access\_token, access\_token\_secret)

# Create API object

api = tweepy.API(auth)

#api object you create then you take the name of the user and tweet count name= "paulocoelho"

# Number of tweets to pull

name="paulocoelho"

tweetCount =10

#calling the user\_timeline function with our parameters

results=api.user\_timeline (id=name, count=tweetCount)

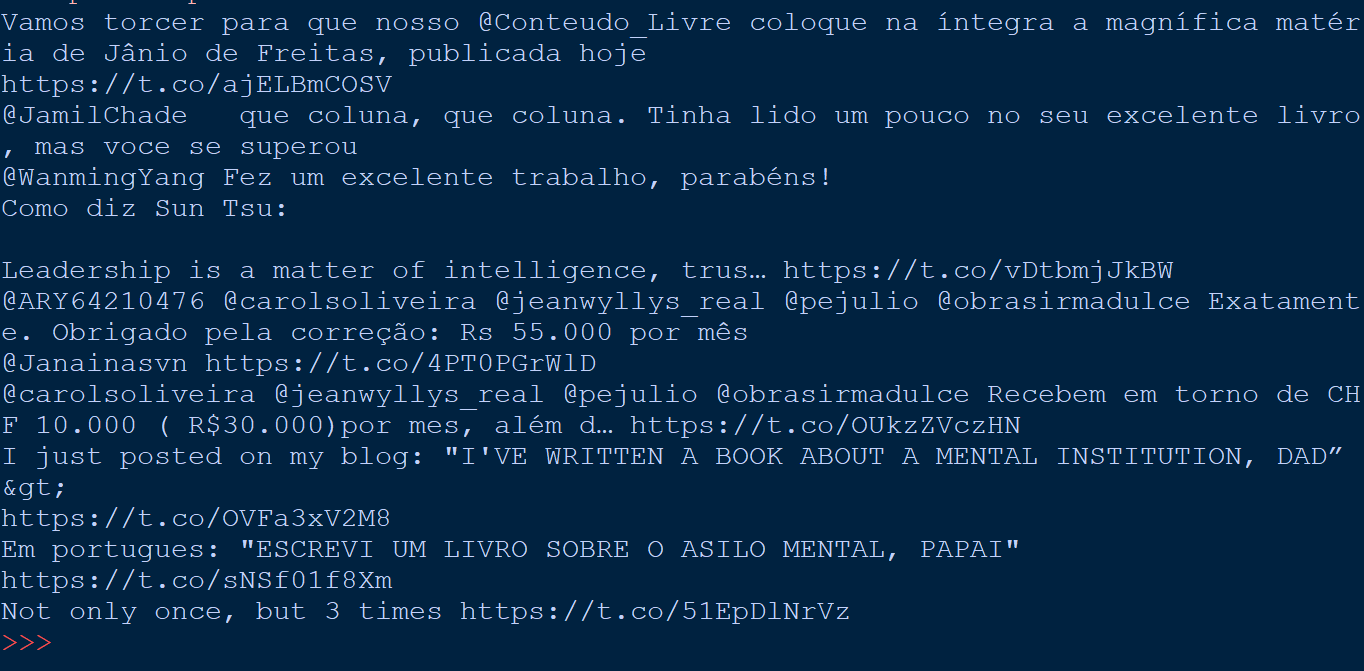
#for each through all tweets pulled

for tweet in results:

#printing the text stored inside the tweet object

print (tweet.text)

**OUTPUT:**



**Practical 8**

**AIM:** Write a program to implement simple web crawler.

**WRITE UP:**

A Web crawler, sometimes called a spider or spiderbot and often shortened to crawler, is an Internet bot that systematically browses the World Wide Web and that is typically operated by search engines for the purpose of Web indexing (web spidering).

**INPUT:**

import requests

from bs4 import BeautifulSoup

plain\_text=requests.get('https://codewithnick.github.io/').text

s=BeautifulSoup(plain\_text,"html.parser")

for link in s.findAll('img'):

tet=link.getText()

print(tet)

print(link)

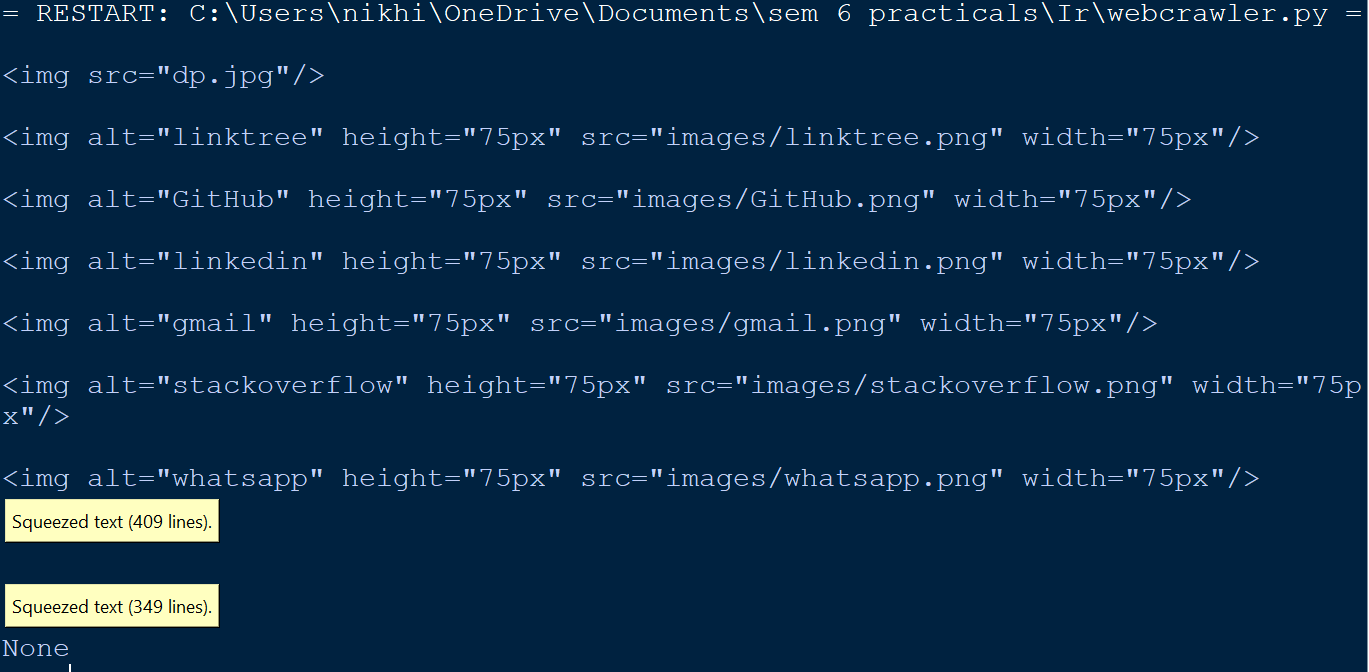
print(plain\_text)

print("")

print(s.text)

print(s.status\_code)

**OUTPUT:**



**Practical 9**

**AIM:** Write a program to parse XML text, generate Web graph and compute topic specific page rank.

**WRITE UP:**

XML: XML stands for eXtensible Markup Language. It was designed to store and transport data. It was designed to be both human- and machine-readable.That’s why, the design goals of XML emphasize simplicity, generality, and usability across the Internet.

The XML file to be parsed in this tutorial is actually a RSS feed.

RSS: RSS(Rich Site Summary, often called Really Simple Syndication) uses a family of standard web feed formats to publish frequently updated informationlike blog entries, news headlines, audio, video. RSS is XML formatted plain text.

The RSS format itself is relatively easy to read both by automated processes and by humans alike.

The RSS processed in this tutorial is the RSS feed of top news stories from a popular news website. You can check it out here. Our goal is to process this RSS feed (or XML file) and save it in some other format for future use.

**INPUT:**

#Python code to illustrate parsing of XML files

# importing the required modules

import csv

import requests

import xml.etree.ElementTree as ET

def loadRSS():

# url of rss feed

url = 'http://wwwnc.cdc.gov/eid/rss/ahead-of-print.xml'

# creating HTTP response object from given url

resp = requests.get(url)

# saving the xml file

with open('topnewsfeed.xml', 'wb') as f:

f.write(resp.content)

def parseXML(xmlfile):

# create element tree object

tree = ET.parse(xmlfile)

# get root element

root = tree.getroot()

# create empty list for news items

newsitems = []

# iterate news items

for item in root.findall('./channel/item'):

# empty news dictionary

news = {}

# iterate child elements of item

for child in item:

# special checking for namespace object content:media

if child.tag == '{http://search.yahoo.com/mrss/}content':

news['media'] = child.attrib['url']

else:

news[child.tag] = child.text.encode('utf8')

# append news dictionary to news items list

newsitems.append(news)

# return news items list

return newsitems

def savetoCSV(newsitems, filename):

# specifying the fields for csv file

fields = ['guid', 'title', 'pubDate', 'description', 'link', 'media']

# writing to csv file

with open(filename, 'w') as csvfile:

# creating a csv dict writer object

writer = csv.DictWriter(csvfile, fieldnames = fields)

# writing headers (field names)

writer.writeheader()

# writing data rows

writer.writerows(newsitems)

def main():

# load rss from web to update existing xml file

loadRSS()

# parse xml file

newsitems = parseXML('topnewsfeed.xml')

# store news items in a csv file

savetoCSV(newsitems, 'topnews.csv')

if \_\_name\_\_ == "\_\_main\_\_":

# calling main function

main()

**OUTPUT:**

