Medicinal Search Engine

This project has been written as the CS F469 Information Retrieval Assignment 1. The project implements basic search engine using inverted index of the medicine data scraped from www.drugs.com. We could theoretically expand the codebase to search through any similar database with relative ease.

Team:

1. **Prateek Agarwal** - 2017A7PS0075H
2. **Shreeya Nelekar** - 2017A7PS0093H
3. **Shriya Choudhary** - 2017AAPS0409H
4. **Shubham Singhal** - 2017AAPS1741H

Usage:

1. Run `python get_med_names.py` :This code scrapes medicine names and corresponding urls from the website.
2. Run `python  med_details.py` :This code scrapes the information of every medicine scraped by get_med_names.py and stores it as json objects in MongoDB.
3. Run `python index_generator.py` : This code generates inverted index of the dataset scraped and also generates tf_idf.p document.
4. Run `python search_query.py` : This code runs the Search Script which takes search query as Command line input and returns the list of medicines corresponding to the search query.

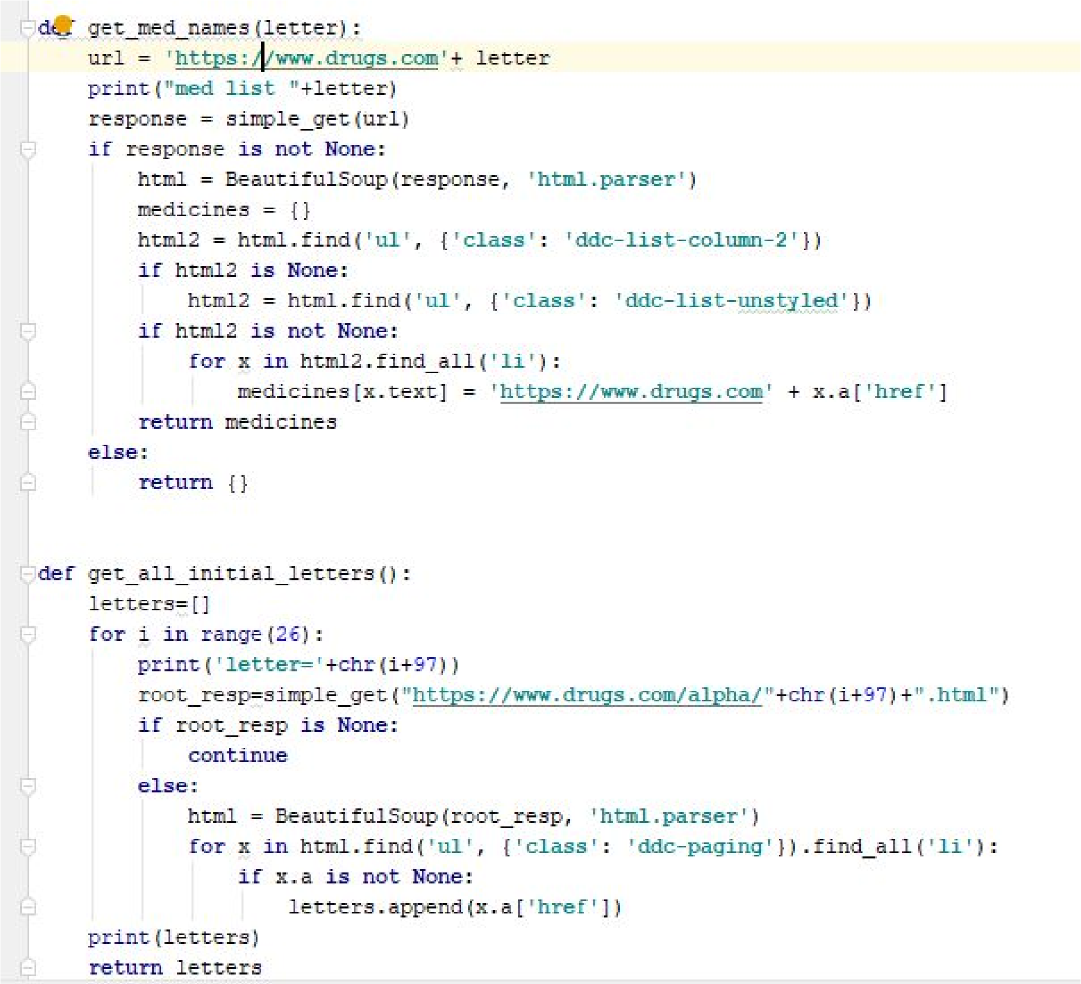
Features:

Following is the list of features we support:

1. Case-insensitive Search Results.
2. Ranking of Search Results.
3. Scraping of over 20000 medicine data

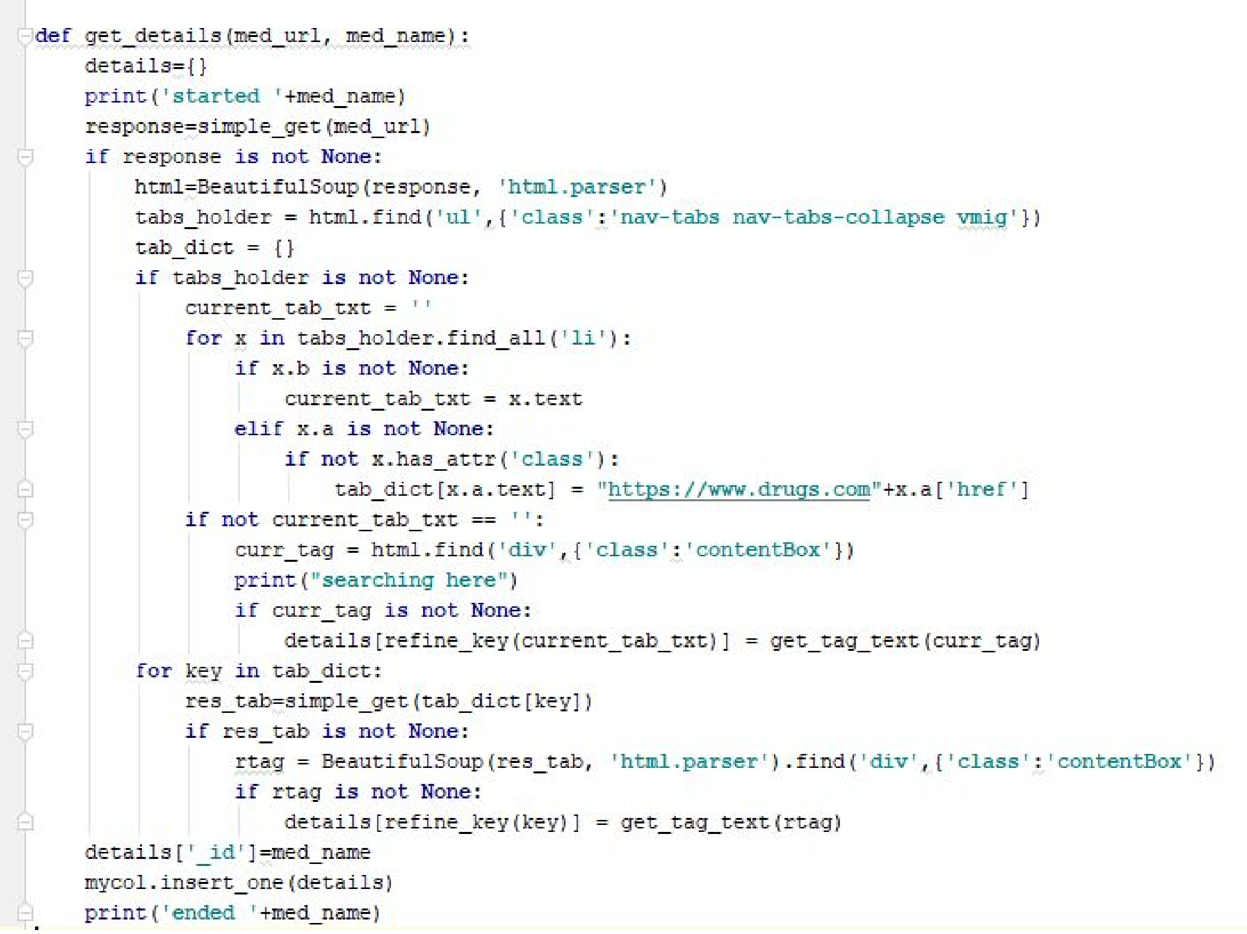
Code Snippets:

1. Get_med_details:



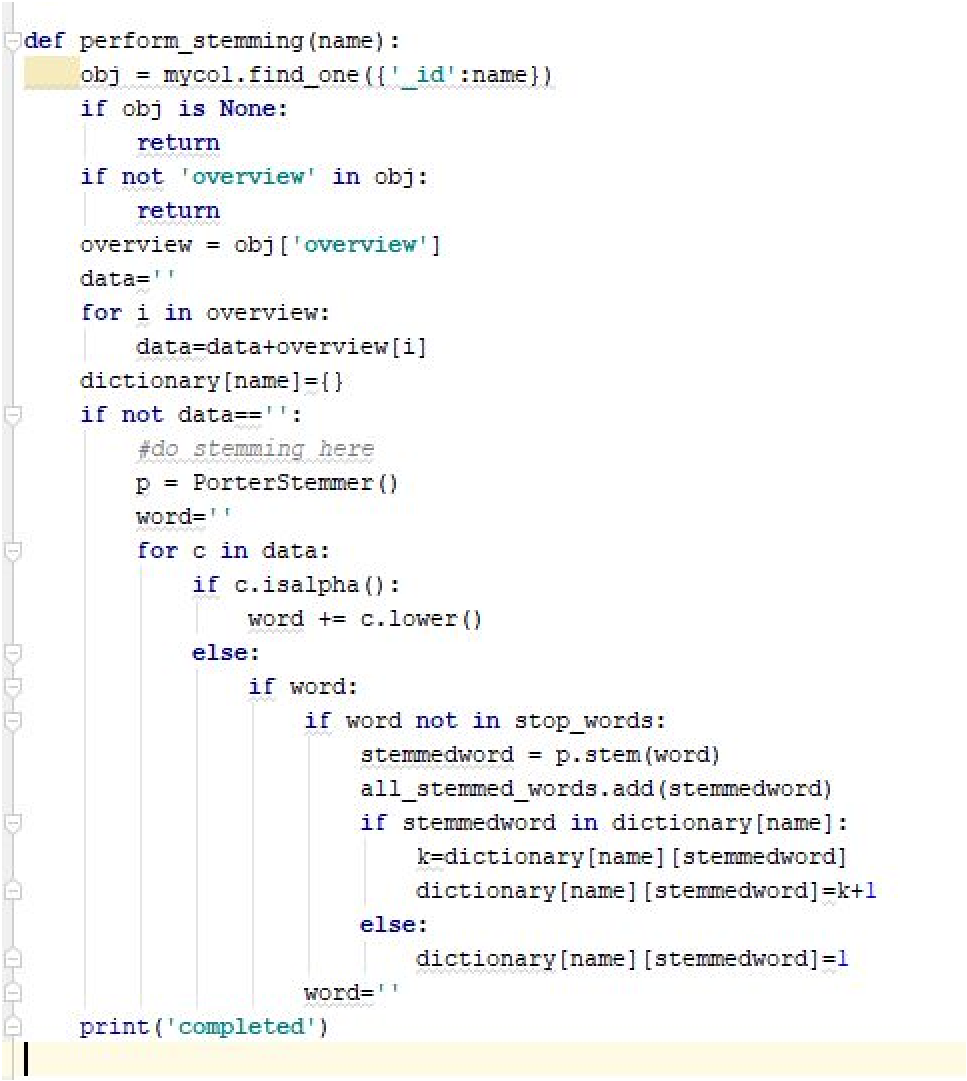
```
def get_med_names(letter):
    url = 'https://www.drugs.com'+ letter
    print("med list "+letter)
    response = simple_get(url)
    if response is not None:
        html = BeautifulSoup(response, 'html.parser')
        medicines = {}
        html2 = html.find('ul', {'class': 'ddc-list-column-2'})
        if html2 is None:
            html2 = html.find('ul', {'class': 'ddc-list-unstyled'})
        if html2 is not None:
            for x in html2.find_all('li'):
                medicines[x.text] = 'https://www.drugs.com' + x.a['href']
        return medicines
    else:
        return {}


def get_all_initial_letters():
    letters=[]
    for i in range(26):
        print('letter='+chr(i+97))
        root_resp=simple_get("https://www.drugs.com/alpha/"+chr(i+97)+".html")
        if root_resp is None:
            continue
        else:
            html = BeautifulSoup(root_resp, 'html.parser')
            for x in html.find('ul', {'class': 'ddc-paging'}).find_all('li'):
                if x.a is not None:
                    letters.append(x.a['href'])
    print(letters)
    return letters
```

2. Med_details :



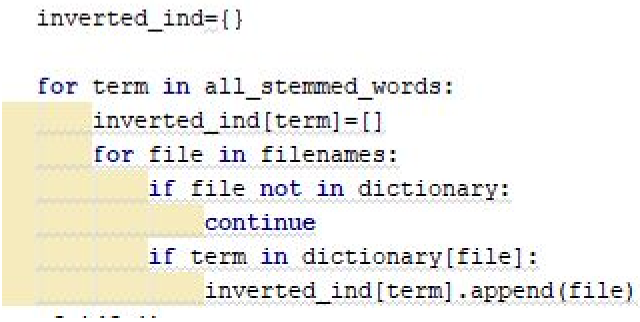
```
def get_details(med_url, med_name):
    details={}
    print('started '+med_name)
    response=simple_get(med_url)
    if response is not None:
        html=BeautifulSoup(response, 'html.parser')
        tabs_holder = html.find('ul',{'class':'nav-tabs nav-tabs-collapse vmig'})
        tab_dict = {}
        if tabs_holder is not None:
            current_tab_txt = ''
            for x in tabs_holder.find_all('li'):
                if x.b is not None:
                    current_tab_txt = x.text
                elif x.a is not None:
                    if not x.has_attr('class'):
                        tab_dict[x.a.text] = "https://www.drugs.com"+x.a['href']
            if not current_tab_txt == '':
                curr_tag = html.find('div',{'class':'contentBox'})
                print("searching here")
                if curr_tag is not None:
                    details[refine_key(current_tab_txt)] = get_tag_text(curr_tag)
        for key in tab_dict:
            res_tab=simple_get(tab_dict[key])
            if res_tab is not None:
                rtag = BeautifulSoup(res_tab, 'html.parser').find('div',{'class':'contentBox'})
                if rtag is not None:
                    details[refine_key(key)] = get_tag_text(rtag)
    details['_id']=med_name
    mycol.insert_one(details)
    print('ended '+med_name)
```

3. Index-generator :
   The core index_generator code that parses and inserts each word in each file into the database. We stem each word using `nltk` before adding to the database. If a word already exists, we increment it's count for that particular file. For this, we are using a 2d dictionary object to store the data.



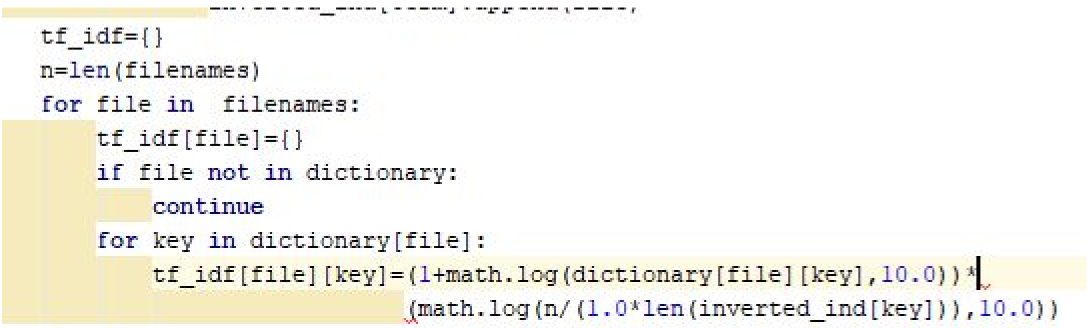
```
def perform_stemming(name):
    obj = mycol.find_one({'_id':name})
    if obj is None:
        return
    if not 'overview' in obj:
        return
    overview = obj['overview']
    data=''
    for i in overview:
        data=data+overview[i]
    dictionary[name]={}
    if not data=='':
        #do stemming here
        p = PorterStemmer()
        word=''
        for c in data:
            if c.isalpha():
                word += c.lower()
            else:
                if word:
                    if word not in stop_words:
                        stemmedword = p.stem(word)
                        all_stemmed_words.add(stemmedword)
                        if stemmedword in dictionary[name]:
                            k=dictionary[name][stemmedword]
                            dictionary[name][stemmedword]=k+1
                        else:
                            dictionary[name][stemmedword]=1
                    word=''
    print('completed')
```

Code to construct an inverted index for each word after stemming:



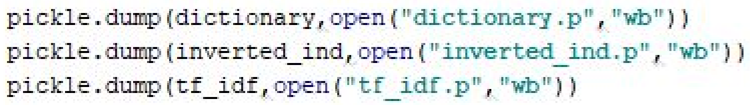
```
inverted_ind={}

for term in all_stemmed_words:
    inverted_ind[term]=[]
    for file in filenames:
        if file not in dictionary:
            continue
        if term in dictionary[file]:
            inverted_ind[term].append(file)
```

Code to calculate TF-IDF for the dataset:



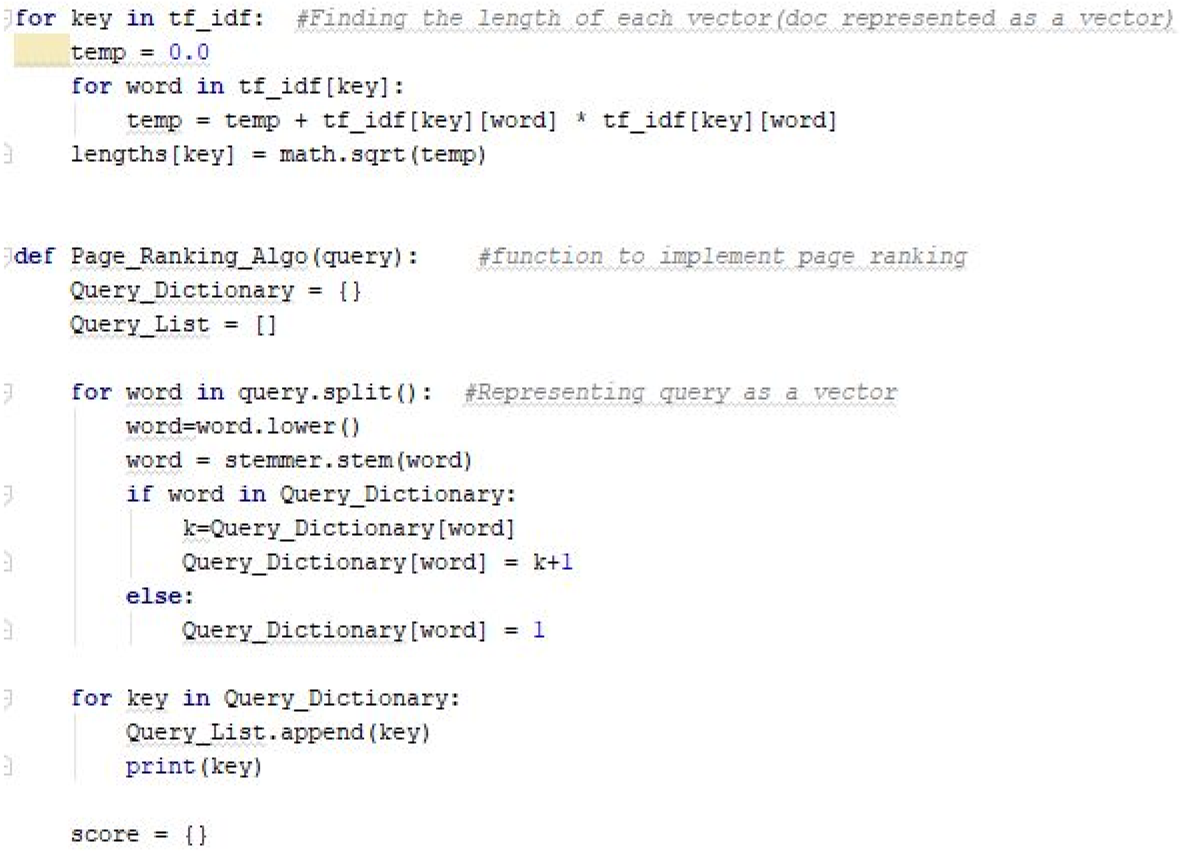
```
tf_idf={}
n=len(filenames)
for file in  filenames:
    tf_idf[file]={}
    if file not in dictionary:
        continue
    for key in dictionary[file]:
        tf_idf[file][key]=(1+math.log(dictionary[file][key],10.0))*
                          (math.log(n/(1.0*len(inverted_ind[key])),10.0))
```

The code to dump the generated dataset for use by the querying GUI:



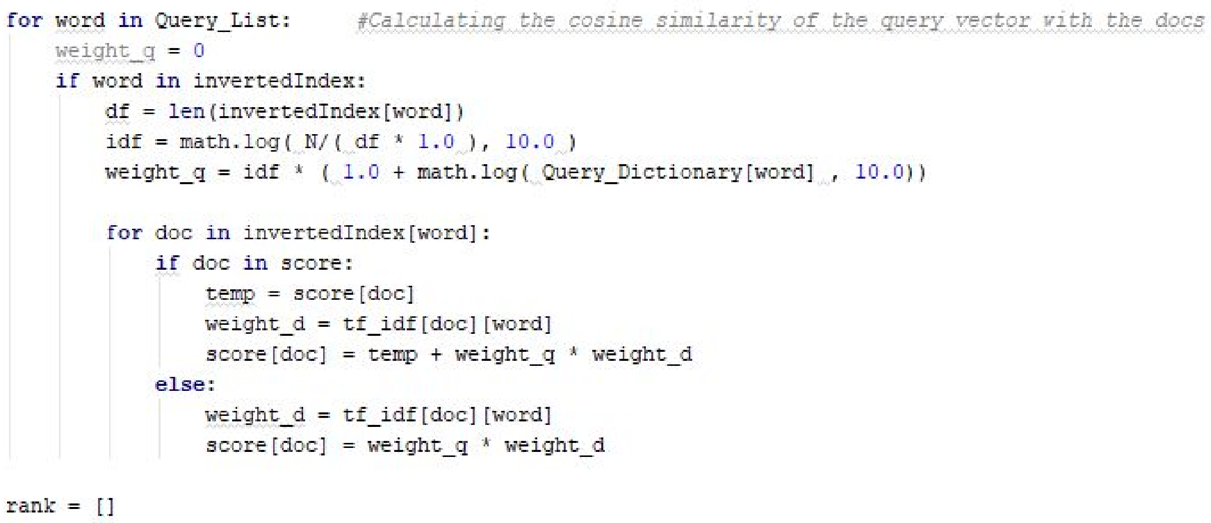
```
pickle.dump(dictionary,open("dictionary.p","wb"))
pickle.dump(inverted_ind,open("inverted_ind.p","wb"))
pickle.dump(tf_idf,open("tf_idf.p","wb"))
```

4. Search_query :
   This is the function that calculates the search results and returns for rendering by the GUI. We query from the inverted index for their presence in the text. After that, sort to return the top 10 results.



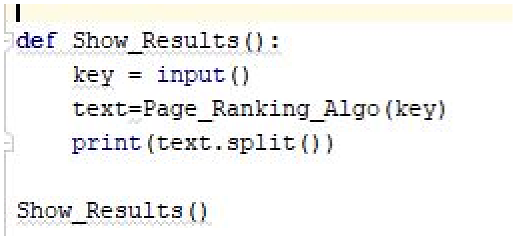
```
for key in tf_idf:  #Finding the length of each vector(doc represented as a vector)
    temp = 0.0
    for word in tf_idf[key]:
        temp = temp + tf_idf[key][word] * tf_idf[key][word]
    lengths[key] = math.sqrt(temp)


def Page_Ranking_Algo(query):    #function to implement page ranking
    Query_Dictionary = {}
    Query_List = []

    for word in query.split():  #Representing query as a vector
        word=word.lower()
        word = stemmer.stem(word)
        if word in Query_Dictionary:
            k=Query_Dictionary[word]
            Query_Dictionary[word] = k+1
        else:
            Query_Dictionary[word] = 1

    for key in Query_Dictionary:
        Query_List.append(key)
        print(key)

    score = {}
```

Getting ranks:



```
for word in Query_List:     #Calculating the cosine similarity of the query vector with the docs
    weight_q = 0
    if word in invertedIndex:
        df = len(invertedIndex[word])
        idf = math.log( N/( df * 1.0 ), 10.0 )
        weight_q = idf * ( 1.0 + math.log( Query_Dictionary[word] , 10.0))

        for doc in invertedIndex[word]:
            if doc in score:
                temp = score[doc]
                weight_d = tf_idf[doc][word]
                score[doc] = temp + weight_q * weight_d
            else:
                weight_d = tf_idf[doc][word]
                score[doc] = weight_q * weight_d

rank = []
```

Shows results:



```
def Show_Results():
    key = input()
    text=Page_Ranking_Algo(key)
    print(text.split())

Show_Results()
```