SIMPLE SEARCH ENGINE

CI213 – INTELLIGENT SYSTEMS

SHANE SMITH

STUDENT ID: 14809022

Table of Contents

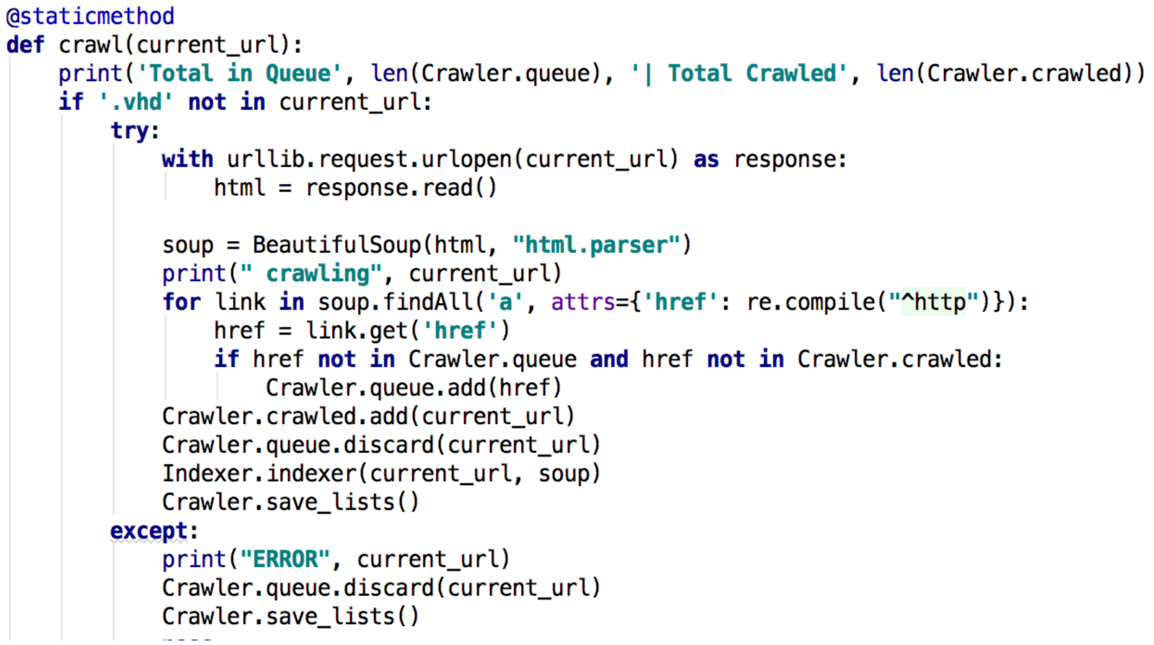
Introduction

In this Assignment we create a prototype Search Engine, this consisted of a Crawler, Indexer and some type of Search Interface. A crawler fetches pages across the internet getting their links within that page. An indexer harvests the pages of the crawler and indexes them in some way, generally allowing a search term/ Keywords which you enter and return the pages which contain them words, preferable in some sort of most relevant order. The GUI is just the search interface which wraps up programmes output in a simple easy to use interface return the results without needing the use of the console. I deciding to use the coding language python for the Search Engine since it has a nice variety of libraries based around Search Engines and it is a nice simple language which I have never used before so it will be a nice learning curve as well.

Report

Crawler

I tried out many crawlers using different libraries like urllibs, beautifulsoup, html parser etc. I ended up just using urllibs.request and bs4, BeautifulSoup.

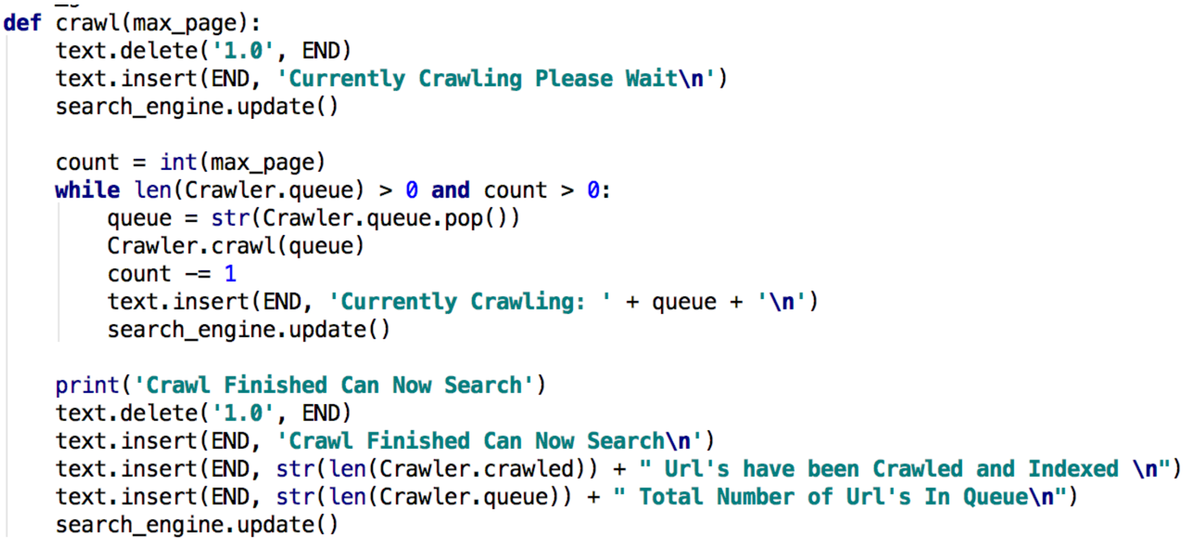


```
@staticmethod
def crawl(current_url):
    print('Total in Queue', len(Crawler.queue), '| Total Crawled', len(Crawler.crawled))
    if '.vhd' not in current_url:
        try:
            with urllib.request.urlopen(current_url) as response:
                html = response.read()

            soup = BeautifulSoup(html, "html.parser")
            print(" crawling", current_url)
            for link in soup.findAll('a', attrs={'href': re.compile("^http")}):
                href = link.get('href')
                if href not in Crawler.queue and href not in Crawler.crawled:
                    Crawler.queue.add(href)
            Crawler.crawled.add(current_url)
            Crawler.queue.discard(current_url)
            Indexer.indexer(current_url, soup)
            Crawler.save_lists()
        except:
            print("ERROR", current_url)
            Crawler.queue.discard(current_url)
            Crawler.save_lists()
```

The above method: crawl(current_url), is my main crawl method which gets all the links from the current URL supplied to it. I first check that the string '.vhd' is not in the current url since I actually tried to download a 3gb file from the url; Whoops, so I added this in to prevent that. I then use urllibs.request to request the url and then I read the html of that url by using read(). The last set up part of the crawler was to convert the html to a BeautifulSoup file to make this next step a lot easier.

I Then make link for each link in the web page by using the function findAll and then adding parameters 'a' to only find a tags and then only getting the attributes 'href' from that a tag. I then get the href from the link and then make sure that It isn't already in the Queued or Crawled sets. I then add the href to the queue set and then add the Current Url; The one we started at, into the crawled set and remove it from the queue set since we don't want to crawl it again. Once I have got all my links I then call the indexer, but we will get to that later on. I then update the Crawled and Queued files. I won’t go into detail of these since they are just general reading and writing to files.

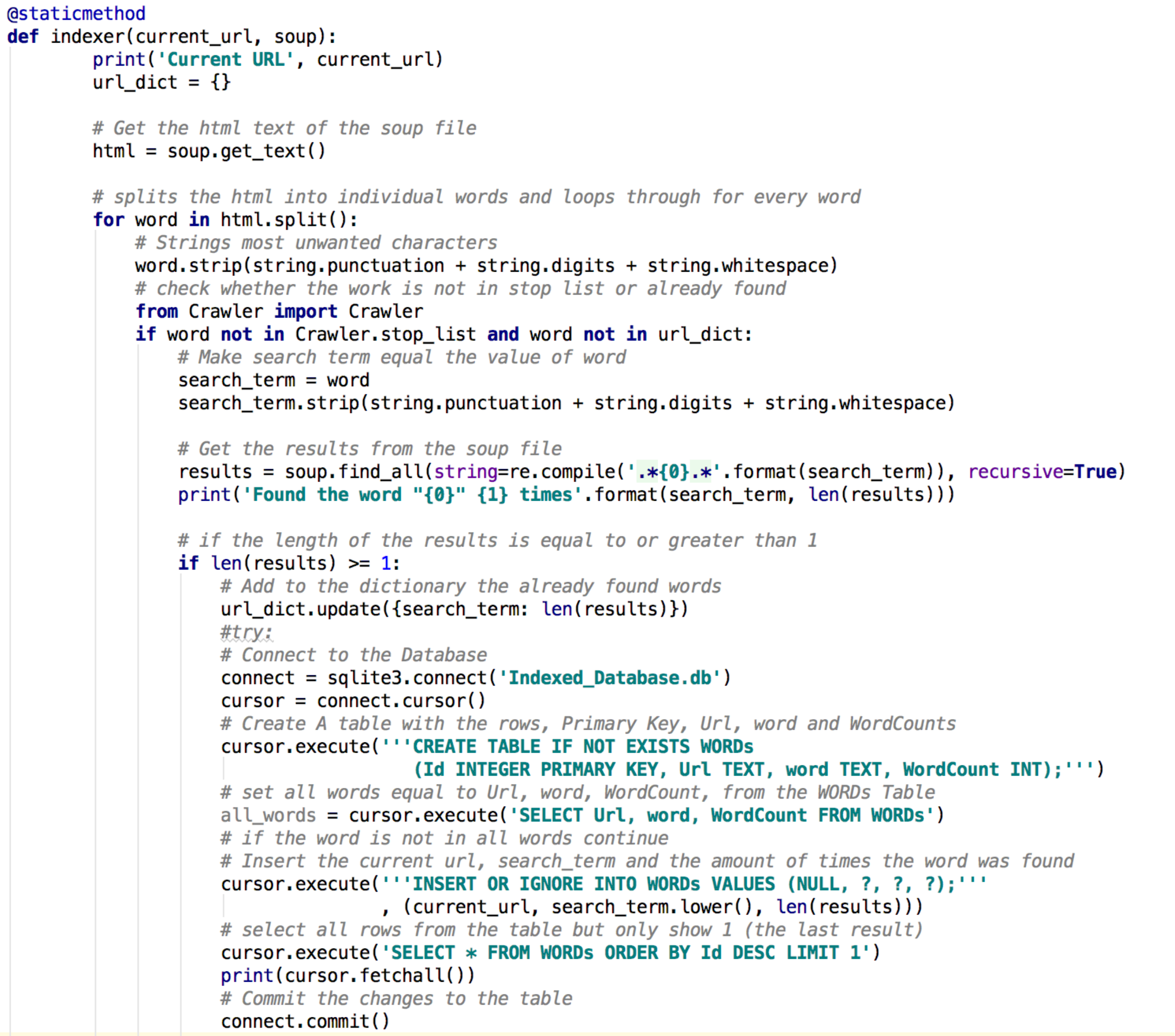
I call the above method from the GUI interface which calls this method:



```
def crawl(max_page):
    text.delete('1.0', END)
    text.insert(END, 'Currently Crawling Please Wait\n')
    search_engine.update()

    count = int(max_page)
    while len(Crawler.queue) > 0 and count > 0:
        queue = str(Crawler.queue.pop())
        Crawler.crawl(queue)
        count -= 1
        text.insert(END, 'Currently Crawling: ' + queue + '\n')
        search_engine.update()

    print('Crawl Finished Can Now Search')
    text.delete('1.0', END)
    text.insert(END, 'Crawl Finished Can Now Search\n')
    text.insert(END, str(len(Crawler.crawled)) + " Url's have been Crawled and Indexed \n")
    text.insert(END, str(len(Crawler.queue)) + " Total Number of Url's In Queue\n")
    search_engine.update()
```

What this method does is gets the maximum number of pages; which is entered by the user. I set count equal to the integer value of max_page and then only queue while the queue set and count are both greater than 0. If they are I then make queue equal to a string value of the first value in the queue set. I then call the crawl(current_url) method which we spoke about above. I repeat this until the while loop doesn’t meet one of the two parameters. All of the ‘text.’ is for the GUI to print the appropriate data to the user.

Indexer



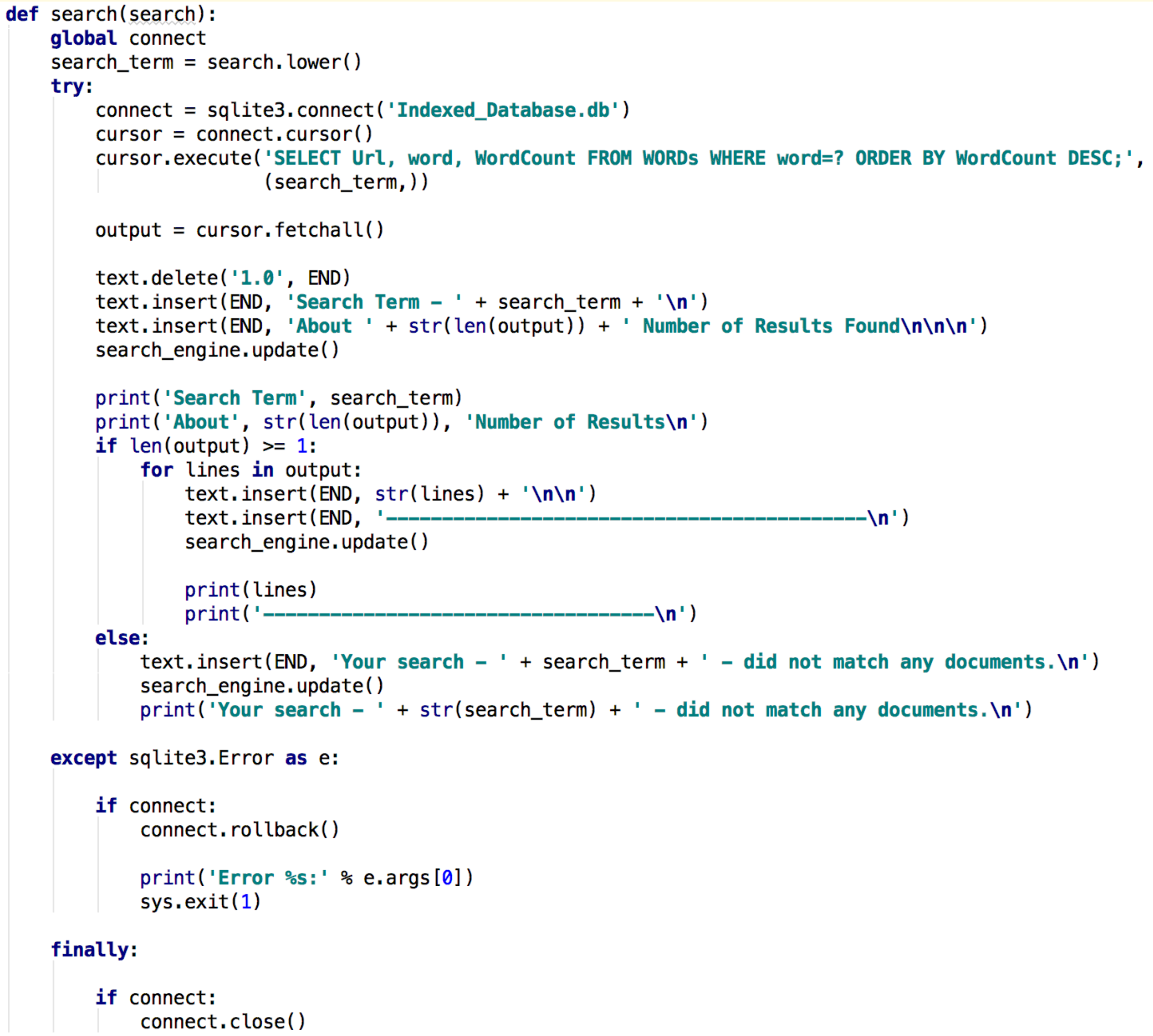
```
@staticmethod
def indexer(current_url, soup):
        print('Current URL', current_url)
        url_dict = {}

        # Get the html text of the soup file
        html = soup.get_text()

        # splits the html into individual words and loops through for every word
        for word in html.split():
            # Strings most unwanted characters
            word.strip(string.punctuation + string.digits + string.whitespace)
            # check whether the work is not in stop list or already found
            from Crawler import Crawler
            if word not in Crawler.stop_list and word not in url_dict:
                # Make search term equal the value of word
                search_term = word
                search_term.strip(string.punctuation + string.digits + string.whitespace)

                # Get the results from the soup file
                results = soup.find_all(string=re.compile('.*{0}.*'.format(search_term)), recursive=True)
                print('Found the word "{0}" {1} times'.format(search_term, len(results)))

                # if the length of the results is equal to or greater than 1
                if len(results) >= 1:
                    # Add to the dictionary the already found words
                    url_dict.update({search_term: len(results)})
                    #try:
                    # Connect to the Database
                    connect = sqlite3.connect('Indexed_Database.db')
                    cursor = connect.cursor()
                    # Create A table with the rows, Primary Key, Url, word and WordCounts
                    cursor.execute('''CREATE TABLE IF NOT EXISTS WORDs
                                      (Id INTEGER PRIMARY KEY, Url TEXT, word TEXT, WordCount INT);''')
                    # set all words equal to Url, word, WordCount, from the WORDs Table
                    all_words = cursor.execute('SELECT Url, word, WordCount FROM WORDs')
                    # if the word is not in all words continue
                    # Insert the current url, search_term and the amount of times the word was found
                    cursor.execute('''INSERT OR IGNORE INTO WORDs VALUES (NULL, ?, ?, ?);'''
                                   , (current_url, search_term.lower(), len(results)))
                    # select all rows from the table but only show 1 (the last result)
                    cursor.execute('SELECT * FROM WORDs ORDER BY Id DESC LIMIT 1')
                    print(cursor.fetchall())
                    # Commit the changes to the table
                    connect.commit()
```

The above method indexer(current_url, soup), is my indexer what the function for the indexer is to get all the words from all the URL's I parse into it and store each word into a database. I first get all the text of the soup file parsed in which is the BeautifulSoup version of the html of the current_url parsed in. I then create a for loop which splits all the text into separate words and I call the value of each word, respectfully named word.

I then for each word in the file strip the punctuation, digit and white space from the ends of it, this semi works since if it has lots of punctuation for example it only strips the ends off. I then make sure that the word we have is not in the stop_list; a bunch of words and characters that I do not want to appear in my results e.g. the, or, and etc. and I make sure that the word is not in url_dict; which is a python dictionary with for each instance of a url stores all the words within in when I come across each word this means that if the word Shane appeared five times once then it for some reason finds Shane five more times then it doesn't add it to the database.

Once I have determined that I haven't found the word already and its not in my stop_list; which I need to still expand the stop words in it. I then check how many times that word appears in the url by using the find_all function with the parameters a string with the format and the value of the search_term which is equal to the current word we found on the webpage. Once I got the word and how many times it appears on the webpage I then add it to the database file; you can see what it is doing by reading the comments I made in the text. I learned how to use sqlite3 mostly by using this website: http://www.tutorialspoint.com/sqlite/sqlite_python.htm

Once the program has finished crawling and indexing the URL's I then want to search the database for the Keywords, I do this by calling the below method



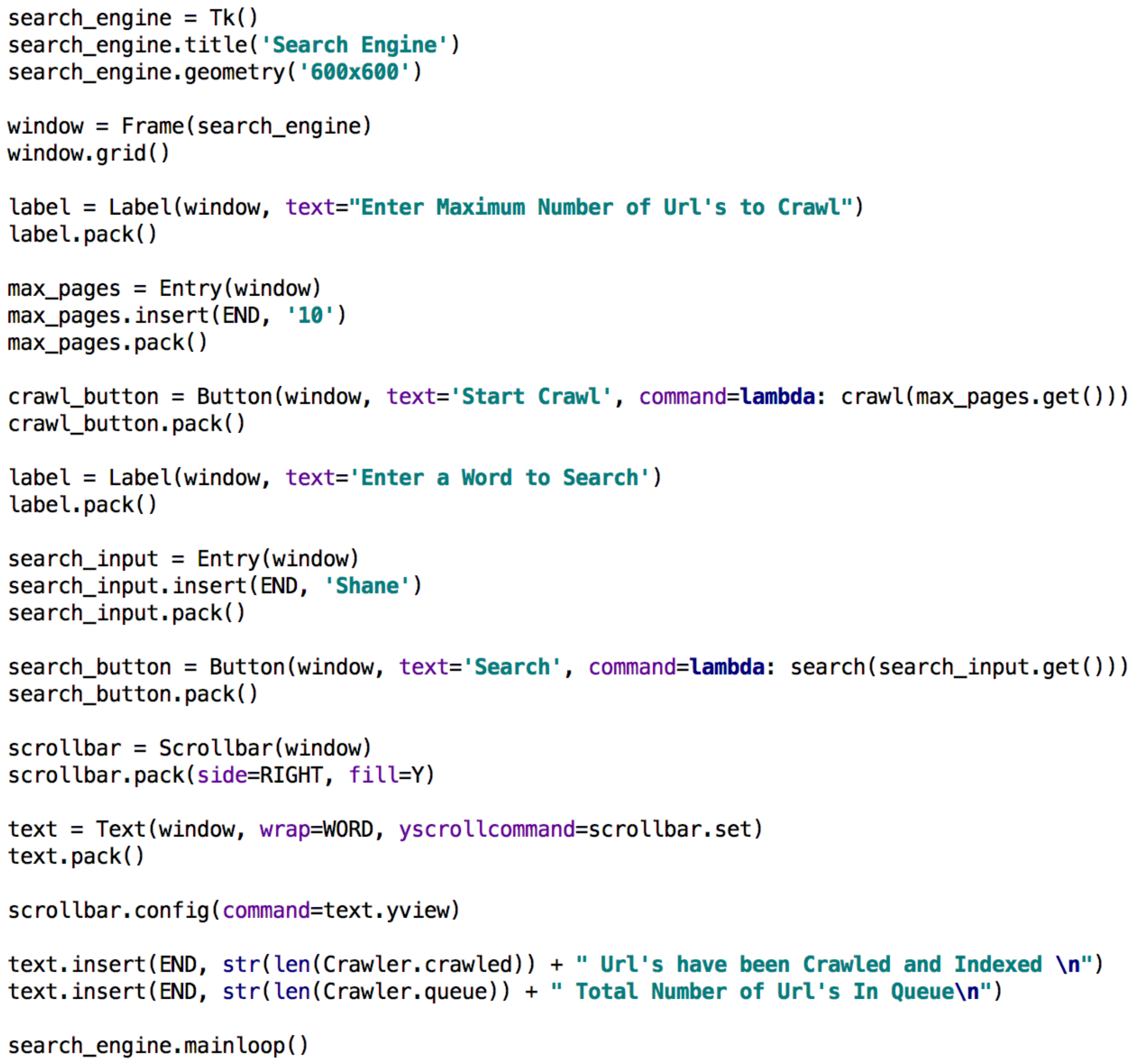
```
def search(search):
    global connect
    search_term = search.lower()
    try:
        connect = sqlite3.connect('Indexed_Database.db')
        cursor = connect.cursor()
        cursor.execute('SELECT Url, word, WordCount FROM WORDs WHERE word=? ORDER BY WordCount DESC;',
                       (search_term,))

        output = cursor.fetchall()

        text.delete('1.0', END)
        text.insert(END, 'Search Term - ' + search_term + '\n')
        text.insert(END, 'About ' + str(len(output)) + ' Number of Results Found\n\n\n')
        search_engine.update()

        print('Search Term', search_term)
        print('About', str(len(output)), 'Number of Results\n')
        if len(output) >= 1:
            for lines in output:
                text.insert(END, str(lines) + '\n\n')
                text.insert(END, '-------------------------------------------\n')
                search_engine.update()

                print(lines)
                print('-----------------------------------\n')
        else:
            text.insert(END, 'Your search - ' + search_term + ' - did not match any documents.\n')
            search_engine.update()
            print('Your search - ' + str(search_term) + ' - did not match any documents.\n')

    except sqlite3.Error as e:

        if connect:
            connect.rollback()

            print('Error %s:' % e.args[0])
            sys.exit(1)

    finally:

        if connect:
            connect.close()
```

The above method, search(search). Gets the user entered search value and checks to see if it is indexed in the database, if it is then it prints out how many times the word appears within each url, the url it appears in and the word itself. It does this for every URL the word has appeared, and it is sorted by highest word count to lowest. So for example if the word Shane appeared in 3 URL's it then would print out the top result first and then so on.

The first step in this method is to change the value search to lowercase so that all the words we deal with through the database and inputs are all lowercase. I then connect to the database the same as before then I select the 'Url', 'word' and 'WordCount' row from the table 'WORDs', and then set the value of the select to output. I then make sure that the

length of the output is 1 or more this is because if there aren't any values which contains that word then I want to be able to tell the user that the word they search doesn't exist in the database.
I then for each output I run the for loop which prints out the results it found to the terminal and most importantly to the GUI so that the user can see their results.

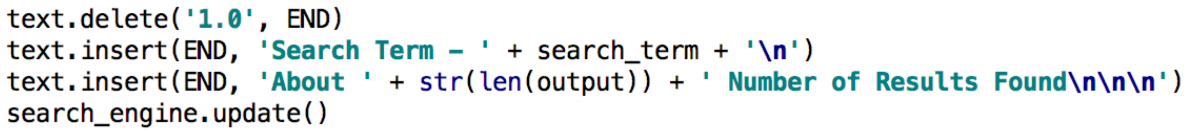
GUI



```
search_engine = Tk()
search_engine.title('Search Engine')
search_engine.geometry('600x600')

window = Frame(search_engine)
window.grid()

label = Label(window, text="Enter Maximum Number of Url's to Crawl")
label.pack()

max_pages = Entry(window)
max_pages.insert(END, '10')
max_pages.pack()

crawl_button = Button(window, text='Start Crawl', command=lambda: crawl(max_pages.get()))
crawl_button.pack()

label = Label(window, text='Enter a Word to Search')
label.pack()

search_input = Entry(window)
search_input.insert(END, 'Shane')
search_input.pack()

search_button = Button(window, text='Search', command=lambda: search(search_input.get()))
search_button.pack()

scrollbar = Scrollbar(window)
scrollbar.pack(side=RIGHT, fill=Y)

text = Text(window, wrap=WORD, yscrollcommand=scrollbar.set)
text.pack()

scrollbar.config(command=text.yview)

text.insert(END, str(len(Crawler.crawled)) + " Url's have been Crawled and Indexed \n")
text.insert(END, str(len(Crawler.queue)) + " Total Number of Url's In Queue\n")

search_engine.mainloop()
```

The above code is most of my GUI code, for the GUI I used tkinter since it was a pre built in library which comes with python, and it has lots of information online to help me learn it.
I Created a simple 600x600 window which opens when you run the program, I then create a frame for the search engine and use the grid() function. I created labels, entry's and buttons for the user to input and use. I Created the labels by using calling the Label() function and assigning it to the window which I defined above. I did the same for buttons except I used the parameter 'command=lambda: "method" '. This meant I was able to call methods that I defined above this which I talked about above. I send in to the methods the value which is currently in the Entry/ text box. I set the default values as 10, since it is a nice small number to crawl and if you do more crawls then it will take longer, generally anything greater than 100 crawls takes quite long and you don't want to be sitting there watching it. I also added a

scrollbar on the text box I created which gets printed to it all the appropriate outputs related to what it is currently doing. So when you start the program it will print out the total number of url crawled and in the queue.



```
text.delete('1.0', END)
text.insert(END, 'Search Term - ' + search_term + '\n')
text.insert(END, 'About ' + str(len(output)) + ' Number of Results Found\n\n\n')
search_engine.update()
```

This is another example where I delete what was last in the text box so we are working on a clean text box. I then print the search term you entered and the total number of results that it found. I call search_engine.update() to update the information currently shown in the search_engine GUI; I named it search_engine I thought it would be most relevant.

Links and How to Run

Have a look at my full repository on github at:

> https://github.com/Gunn3r1995/Search

To Run my code, see link:

> https://github.com/Gunn3r1995/Search/blob/master/README.md

or Just run it like any other python program but remember to install the library's needed.

Reflection

I had a lot of fun creating a Search Engine, there was just something about seeing hundreds of links printing out to the terminal.

Anyway, when I started to create the crawler it nice and easy until I wanted to crawl multiple pages and get more and more URL's then I started running into lots of problems like the link wasn't the full web address it was only a local call from within the website domain and lots of other fun errors to deal with. I think that after completing this and dealing with all the errors has made me really understand what the crawler is actually doing and just how big the internet really is.

The Indexer was a different kind of challenge to the Crawler since now I got all the links what do I do with them, I had multiple different attempts at the Crawler and the Indexer, surprisingly one of the bigger problems I had was when and how to call the Crawler and Indexer, for example should I crawl all the pages and then once I completed all the crawls, then should I Index etc. I ending up Crawling one page getting the all links and then index the current url. Another big problem I had with the Indexer was at first I was using a text file to save the indexed data, this made it really hard to query it since I wanted to get the url, word and word count all in one and it wasn't working it was to much to figure out and to much output to a txt file. Changing over to a database made it really easy to add the indexed data and query the results.

Create the GUI was really nice to play with since it gave the program a nice wrap up to actually feel like a somewhat finished and working program. But it did give it a nice finishing touch and made it a lot easier to use the program.

I think I would improve the crawler a little but by adding a better search algorithm in it to search through all of the website then move onto a different website since at the moment it will crawl the first page then just end up crawling next the link found after it crawled so it could take a lot of time to finish one website since it keeps branching off to other sites.

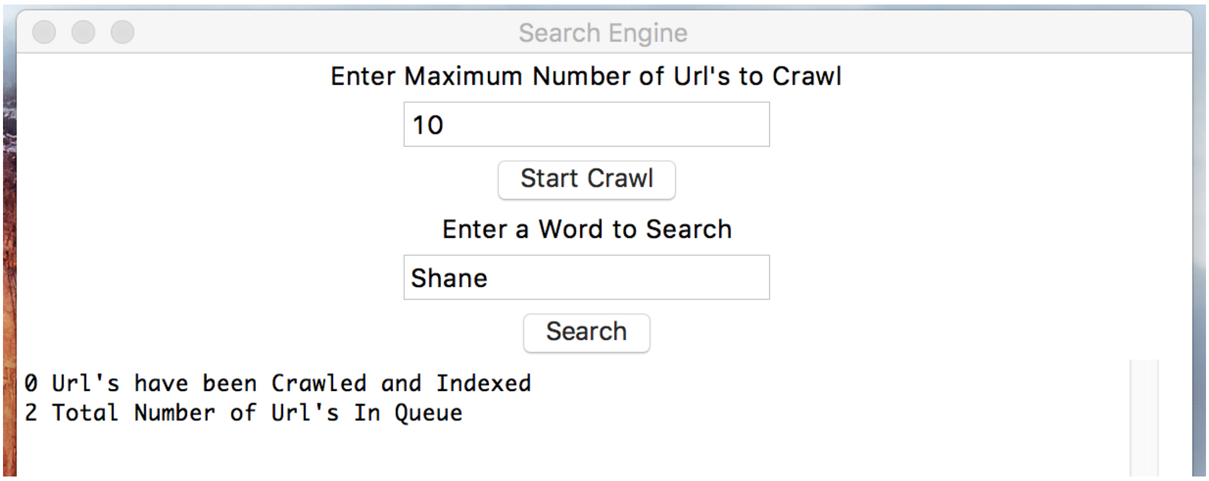
I think the Indexer itself is quite good, I would like improve the stop list functionality of this since a lot of the time I get unwanted words in the database like {san:serif etc. but currently without add each instance into the stop_list it doesn't really filter them out, with some more time on this I believe I could improve this part of the indexer a lot more.

I think the GUI is nice and simple I would like to pair it up with html but I ran out of time. I would like to look into trying to improve the performance of the GUI since at least on my mac It doesn't properly scroll through the currently crawling since there is some performance issues with it, other than that I think it is all that is needed for this program.

Conclusion

To conclude I think I have learnt a lot about how Search Engines work with how they get the links by crawling through pages across the internet, and how they collect the data from the websites to generate search results from a range of different websites. after exams I would like to add my Search engine into my website and use it a custom search engine but only for my domain so I have a fully indexed database of my personal website.

Output Demonstration

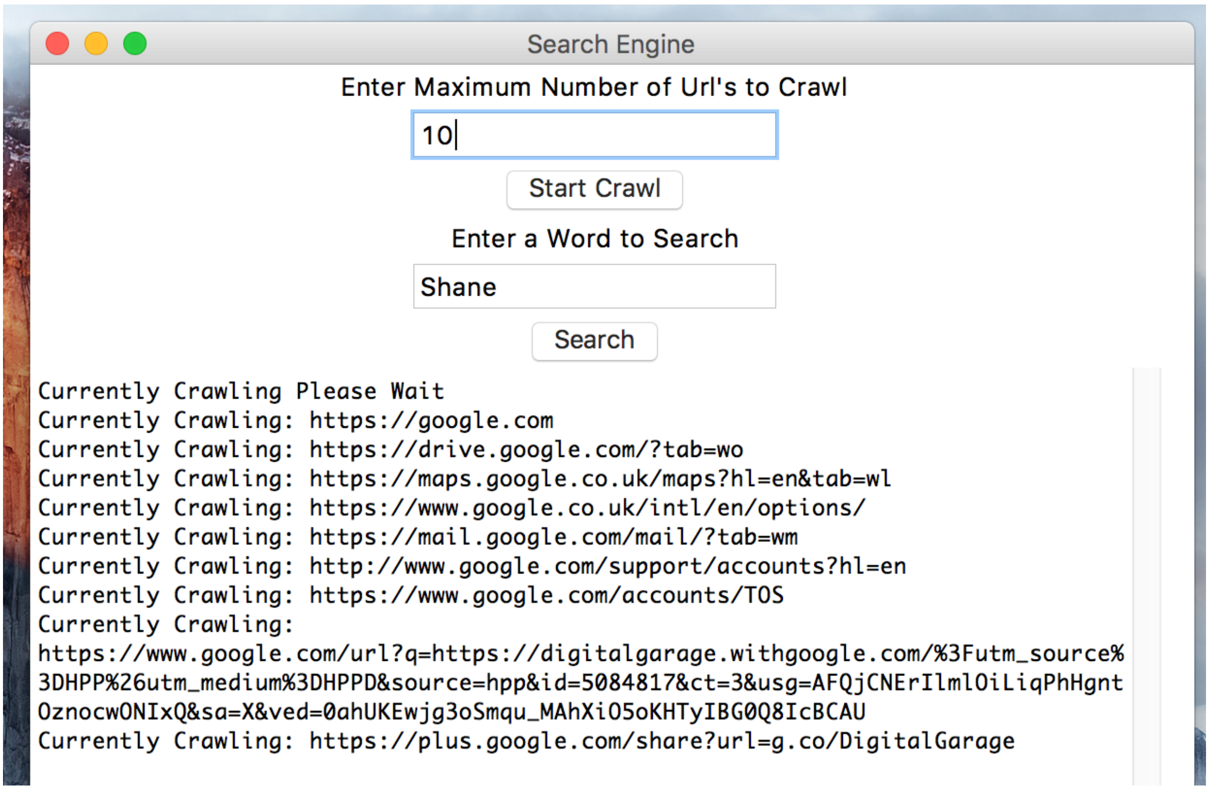


Enter Maximum Number of Url's to Crawl: 10

Enter a Word to Search: Shane

```
0 Url's have been Crawled and Indexed
2 Total Number of Url's In Queue
```

First Time Opening My Program, Has 2 links in queue, https://google.com and http://shanesmithcv.com these are the two I added into the Queue.txt file within the program, Could easy make and input to ask for these values.

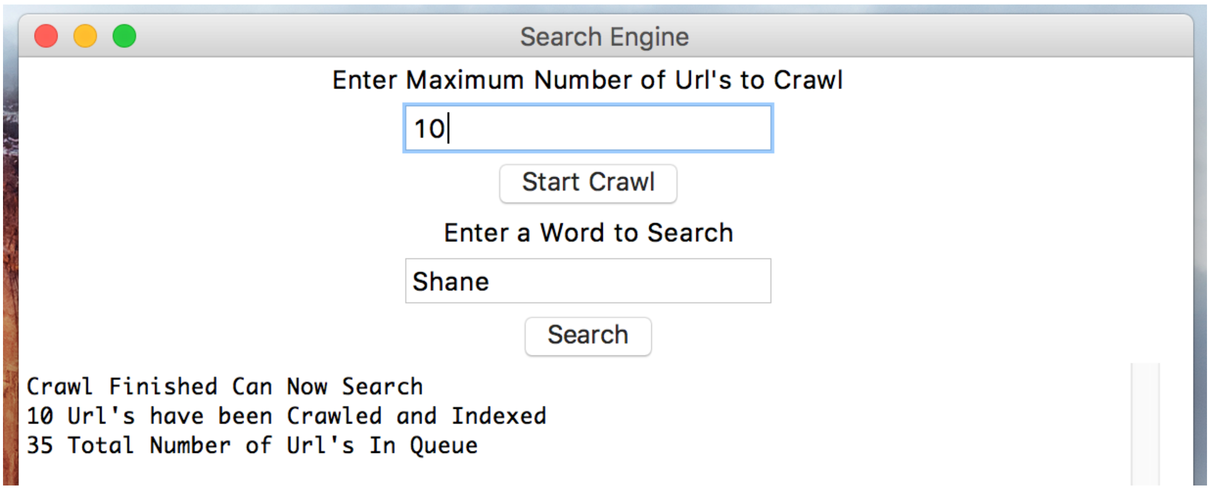


Enter Maximum Number of Url's to Crawl: 10

Enter a Word to Search: Shane

```
Currently Crawling Please Wait
Currently Crawling: https://google.com
Currently Crawling: https://drive.google.com/?tab=wo
Currently Crawling: https://maps.google.co.uk/maps?hl=en&tab=wl
Currently Crawling: https://www.google.co.uk/intl/en/options/
Currently Crawling: https://mail.google.com/mail/?tab=wm
Currently Crawling: http://www.google.com/support/accounts?hl=en
Currently Crawling: https://www.google.com/accounts/TOS
Currently Crawling:
https://www.google.com/url?q=https://digitalgarage.withgoogle.com/%3Futm_source%
3DHPP%26utm_medium%3DHPPD&source=hpp&id=5084817&ct=3&usg=AFQjCNErIlmlOiLiqPhHgnt
OznocwONIxQ&sa=X&ved=0ahUKEwjg3oSmqu_MAhXiO5oKHTyIBG0Q8IcBCAU
Currently Crawling: https://plus.google.com/share?url=g.co/DigitalGarage
```

Currently Crawling Output after pressing Start Crawl with a total of 10 Crawls, (Not all of them listed currently on the Output since I took screen shot before it was finished)



Enter Maximum Number of Url's to Crawl

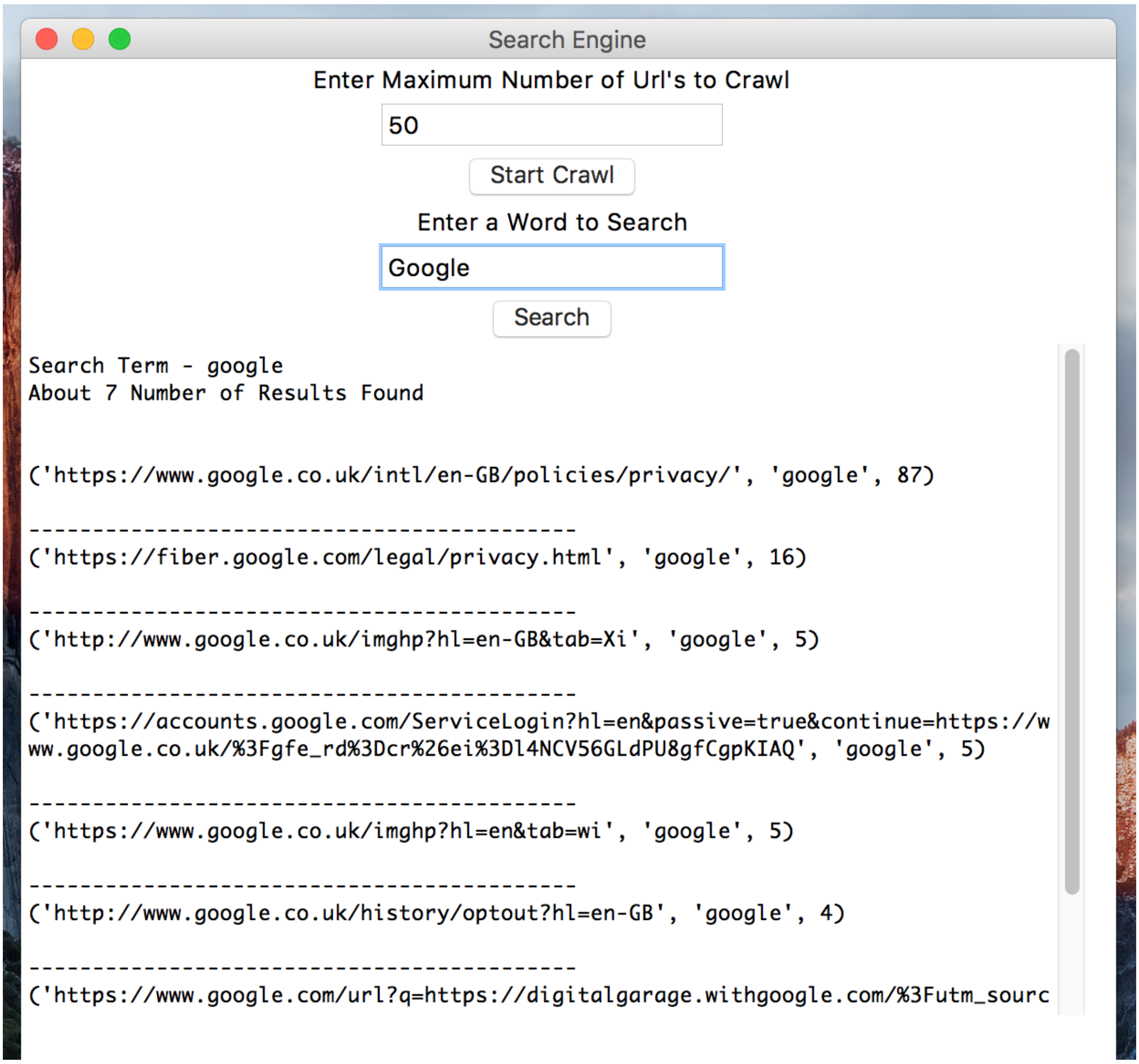
10

Enter a Word to Search

Shane

```
Crawl Finished Can Now Search
10 Url's have been Crawled and Indexed
35 Total Number of Url's In Queue
```

Shows the screen once it has finished with the total of URL's Crawled and total in the Queue.



Enter Maximum Number of Url's to Crawl

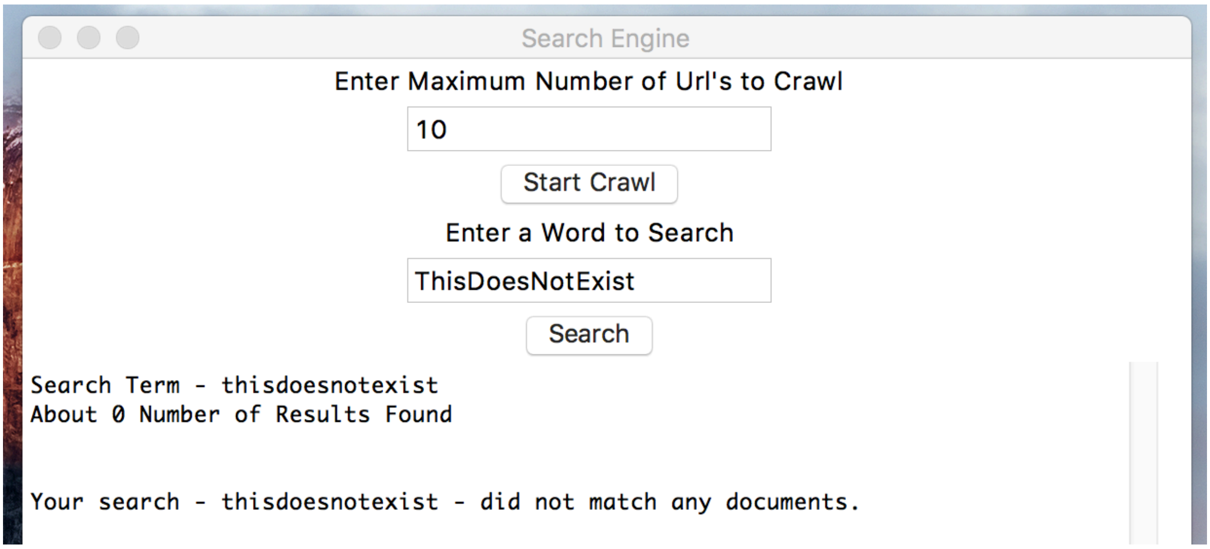
50

Enter a Word to Search

Google

```
Search Term - google
About 7 Number of Results Found


('https://www.google.co.uk/intl/en-GB/policies/privacy/', 'google', 87)

-------------------------------------------
('https://fiber.google.com/legal/privacy.html', 'google', 16)

-------------------------------------------
('http://www.google.co.uk/imghp?hl=en-GB&tab=Xi', 'google', 5)

-------------------------------------------
('https://accounts.google.com/ServiceLogin?hl=en&passive=true&continue=https://w
ww.google.co.uk/%3Fgfe_rd%3Dcr%26ei%3Dl4NCV56GLdPU8gfCgpKIAQ', 'google', 5)

-------------------------------------------
('https://www.google.co.uk/imghp?hl=en&tab=wi', 'google', 5)

-------------------------------------------
('http://www.google.co.uk/history/optout?hl=en-GB', 'google', 4)

-------------------------------------------
('https://www.google.com/url?q=https://digitalgarage.withgoogle.com/%3Futm_sourc
```

Shows the Results with the search term Google. Total number of results is 4 and you can see that it is order by most to least on the word count. Also shows the scroll bar since the rest of the results are further down the page.



```
Search Term - thisdoesnotexist
About 0 Number of Results Found


Your search - thisdoesnotexist - did not match any documents.
```

Shows the screen with the number of results equals 0, so I tell the user that their search for there current search term did not match any documents.

Source Code

Main.py

```
import sqlite3
from tkinter import *
from Crawler import Crawler


def init_gui():
    def crawl(max_page):
        text.delete('1.0', END)
        text.insert(END, 'Currently Crawling Please Wait\n')
        search_engine.update()

        count = int(max_page)
        while len(Crawler.queue) > 0 and count > 0:
            queue = str(Crawler.queue.pop())
            Crawler.crawl(queue)
            count -= 1
            text.insert(END, 'Currently Crawling: ' + queue + '\n')
            search_engine.update()

        print('Crawl Finished Can Now Search')
        text.delete('1.0', END)
        text.insert(END, 'Crawl Finished Can Now Search\n')
        text.insert(END, str(len(Crawler.crawled)) + " Url's have been Crawled and Indexed
\n")
        text.insert(END, str(len(Crawler.queue)) + " Total Number of Url's In Queue\n")
        search_engine.update()

        Crawler.save_lists()

    def search(search):
        global connect
        search_term = search.lower()
        try:
            connect = sqlite3.connect('Indexed_Database.db')
            cursor = connect.cursor()
            cursor.execute('SELECT Url, word, WordCount FROM WORDS WHERE word=? ORDER BY
WordCount DESC;',
                           (search_term,))

            output = cursor.fetchall()

            text.delete('1.0', END)
            text.insert(END, 'Search Term - ' + search_term + '\n')
            text.insert(END, 'About ' + str(len(output)) + ' Number of Results Found\n\n\n')
            search_engine.update()

            print('Search Term', search_term)
            print('About', str(len(output)), 'Number of Results\n')

            if len(output) >= 1:
                for lines in output:
                    text.insert(END, str(lines) + '\n\n')
                    text.insert(END, '------------------------------------------\n')
                    search_engine.update()

                    print(lines)
                    print('----------------------------------\n')
            else:
                text.insert('Your search - ' + str(search_term) + ' - did not match any
documents.\n')
                search_engine.update()
                print('Your search - ', str(search_term), ' - did not match any
documents.\n')

        except sqlite3.Error as e:

            if connect:
                connect.rollback()

                print('Error %s:' % e.args[0])
                sys.exit(1)

        finally:
            if connect:
                connect.close()

    search_engine = Tk()
    search_engine.title('Search Engine')
    search_engine.geometry('600x600')

    window = Frame(search_engine)
    window.grid()

    label = Label(window, text="Enter Maximum Number of Url's to Crawl")
    label.pack()

    max_pages = Entry(window)
    max_pages.insert(END, '10')
    max_pages.pack()

    crawl_button = Button(window, text='Start Crawl', command=lambda: crawl(max_pages.get()))
    crawl_button.pack()

    label = Label(window, text='Enter a Word to Search')
    label.pack()

    search_input = Entry(window)
    search_input.insert(END, 'Shane')
    search_input.pack()

    search_button = Button(window, text='Search', command=lambda: search(search_input.get()))
    search_button.pack()

    scrollbar = Scrollbar(window)
    scrollbar.pack(side=RIGHT, fill=Y)

    text = Text(window, wrap=WORD, yscrollcommand=scrollbar.set)
    text.pack()

    scrollbar.config(command=text.yview)

    text.insert(END, str(len(Crawler.crawled)) + " Url's have been Crawled and Indexed \n")
    text.insert(END, str(len(Crawler.queue)) + " Total Number of Url's In Queue\n")

    search_engine.mainloop()

Crawler.create_file()
init_gui()

Crawler.save_lists()
sys.exit()
```

Crawler.py

```
import re
import os
import urllib.request
from bs4 import BeautifulSoup
from Indexer import Indexer


class Crawler:
    # initialising Class Variables
    stop_list = set()
    queue = set()
    crawled = set()

    @staticmethod
    def crawl(current_url):
        print('Total in Queue', len(Crawler.queue), '| Total Crawled', len(Crawler.crawled))
        if '.vhd' not in current_url:
            try:
                with urllib.request.urlopen(current_url) as response:
                    html = response.read()

                soup = BeautifulSoup(html, "html.parser")
                print(" crawling", current_url)
                for link in soup.findAll('a', attrs={'href': re.compile("^http")}):
                    href = link.get('href')
                    if href not in Crawler.queue and href not in Crawler.crawled:
                        Crawler.queue.add(href)
                Crawler.crawled.add(current_url)
                Crawler.queue.discard(current_url)
                Indexer.indexer(current_url, soup)
                Crawler.save_lists()
            except:
                print("ERROR", current_url)
                Crawler.queue.discard(current_url)
                Crawler.save_lists()
                pass

    @staticmethod
    def save_lists():
        with open('Search/Queue.txt', 'w') as file:
            for link in Crawler.queue:
                file.write(link + '\n')
            file.close()

        with open('Search/Crawled.txt', 'w') as file:
            for link in Crawler.crawled:
                file.write(link + '\n')
            file.close()

    @staticmethod
    def read_file(directory):
        if directory == 'Search/Queue.txt':
            with open('Search/Queue.txt', 'rt') as file:
                    for line in file:
                        Crawler.queue.add(line.replace('\n', ''))
                    file.close()

        if directory == 'Search/Crawled.txt':
            with open('Search/Crawled.txt', 'rt') as file:
                for line in file:
                    Crawler.crawled.add(line.replace('\n', ''))
                file.close()

        if directory == 'Search/stop_word_list.txt':
            with open('Search/stop_word_list.txt', 'rt') as file:
                for line in file:
                    Crawler.stop_list.add(line.replace('\n', ''))
                file.close()

    @staticmethod
    def create_file():
        if not os.path.exists('Search/Queue.txt'):
            file = open('Search/Queue.txt', 'w')
            file.write('http://shanesmithcv.com\n')
            file.write('https://google.com\n')
            file.close()
            Crawler.read_file('Search/Queue.txt')
```

```
else:
    Crawler.read_file('Search/Queue.txt')

if not os .path.isfile('Search/Crawled.txt'):
    file = open('Search/Crawled.txt', 'w')
    file.write('')
    file.close()

    Crawler.read_file('Search/Crawled.txt')
else:
    Crawler.read_file('Search/Crawled.txt')

Crawler.read_file('Search/stop_word_list.txt')
```

Indexer.py

```
import string
import sqlite3
from tkinter import *

class Indexer:

    @staticmethod
    def indexer(current_url, soup):
            print('Current URL', current_url)
            url_dict = {}

            # Get the html text of the soup file
            html = soup.get_text()

            # splits the html into individual words and loops through for every word
            for word in html.split():
                # Strings most unwanted characters
                word.strip(string.punctuation + string.digits + string.whitespace)
                # check whether the work is not in stop list or already found
                from Crawler import Crawler
                if word not in Crawler.stop_list and word not in url_dict:
                    # Make search term equal the value of word
                    search_term = word
                    search_term.strip(string.punctuation + string.digits + string.whitespace)

                    # Get the results from the soup file
                    results = soup.find_all(string=re.compile('.*{0}.*'.format(search_term)),
recursive=True)
                    print('Found the word "{0}" {1} times'.format(search_term, len(results)))

                    # if the length of the results is equal to or greater than 1
                    if len(results) >= 1:
                        # Add to the dictionary the already found words
                        url_dict.update({search_term: len(results)})
                        # Connect to the Database
                        connect = sqlite3.connect('Indexed_Database.db')
                        cursor = connect.cursor()
                        # Create A table with the rows, Primary Key, Url, word and WordCounts
                        cursor.execute('''CREATE TABLE IF NOT EXISTS WORDS
                                          (Id INTEGER PRIMARY KEY, Url TEXT, word TEXT,
WordCount INT);''')
                        # set all words equal to Url, word, WordCount, from the WORDs Table
                        all_words = cursor.execute('SELECT Url, word, WordCount FROM WORDS')
                        # if the word is not in all words continue
                        # Insert the current url, search_term and the amount of times the
word was found
                        cursor.execute('''INSERT OR IGNORE INTO WORDS VALUES (NULL, ?, ?,
?);'''
                                       , (current_url, search_term.lower(), len(results)))
                        # select all rows from the table but only show 1 (the last result)
                        cursor.execute('SELECT * FROM WORDS ORDER BY Id DESC LIMIT 1')
                        print(cursor.fetchall())
                        # Commit the changes to the table
                        connect.commit()
```

Click for stop_list_file:

> https://github.com/Gunn3r1995/Search/blob/master/Search/stop_word_list.txt

Click for Crawled.txt

> https://github.com/Gunn3r1995/Search/blob/master/Search/Crawled.txt

Click for Queue.txt

> https://github.com/Gunn3r1995/Search/blob/master/Search/Queue.txt

Click for Indexed_Database

> https://github.com/Gunn3r1995/Search/blob/master/Indexed_Database.db

See below for example output of Indexed_Database or Download the from link about and open in a database viewer like sqlitebrowser

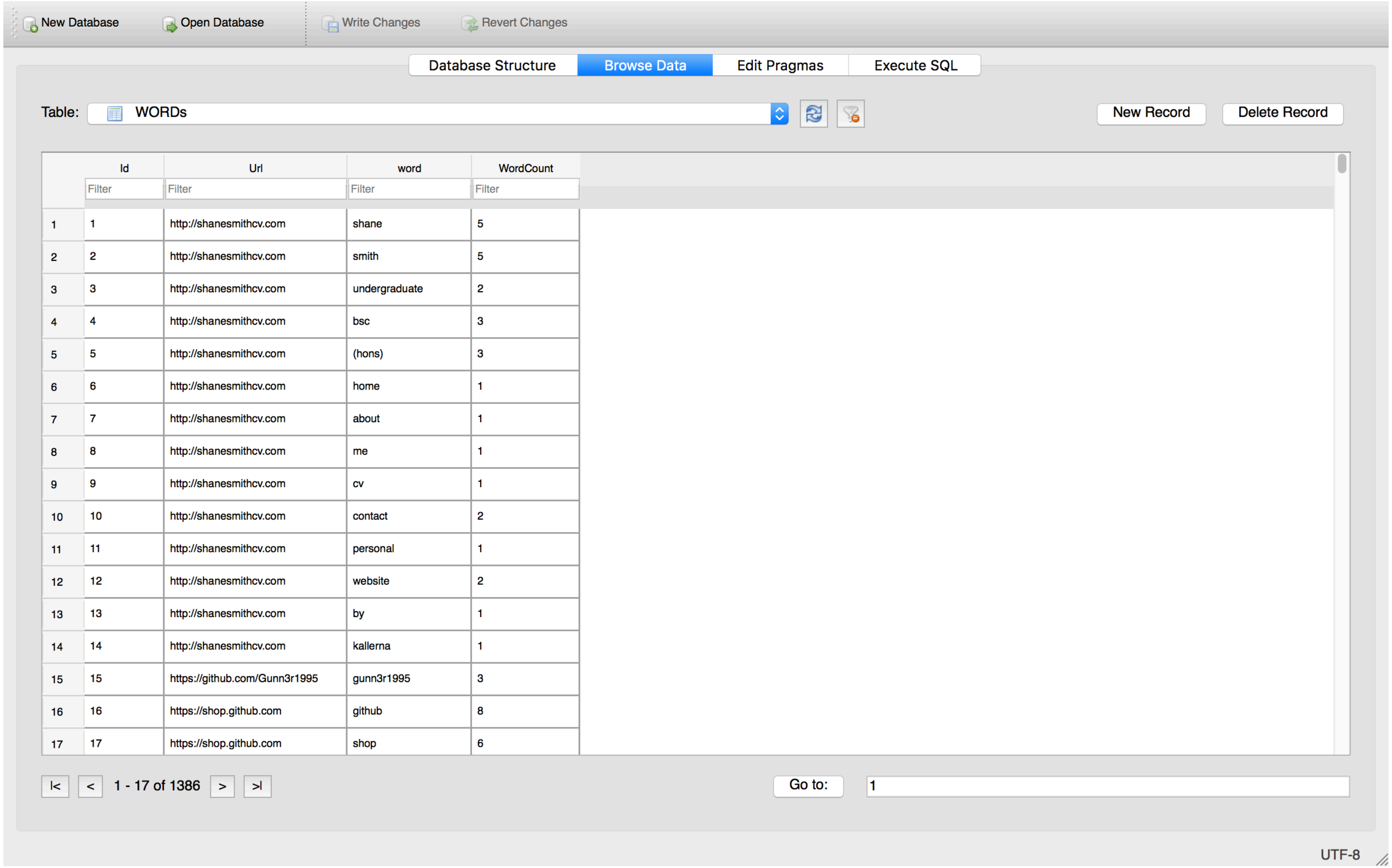


Table: WORDs

	Id	Url	word	WordCount
	Filter	Filter	Filter	Filter
1	1	http://shanesmithcv.com	shane	5
2	2	http://shanesmithcv.com	smith	5
3	3	http://shanesmithcv.com	undergraduate	2
4	4	http://shanesmithcv.com	bsc	3
5	5	http://shanesmithcv.com	(hons)	3
6	6	http://shanesmithcv.com	home	1
7	7	http://shanesmithcv.com	about	1
8	8	http://shanesmithcv.com	me	1
9	9	http://shanesmithcv.com	cv	1
10	10	http://shanesmithcv.com	contact	2
11	11	http://shanesmithcv.com	personal	1
12	12	http://shanesmithcv.com	website	2
13	13	http://shanesmithcv.com	by	1
14	14	http://shanesmithcv.com	kallerna	1
15	15	https://github.com/Gunn3r1995	gunn3r1995	3
16	16	https://shop.github.com	github	8
17	17	https://shop.github.com	shop	6

1 - 17 of 1386

Go to: 1