

1619313040/3301 (Data Science Oriented Programming Language-Python)

SEMESTER 2 (Spring), 2021/2022

INDIVIDUAL FINAL REPORT

NAME: MD Omar Faruq

STUDENT NO.: 201939060073

SUBMITTED TO:

Assoc. Prof. FANG YU

MARKING SCHEME (TO BE FILLED BY THE LECTURER)

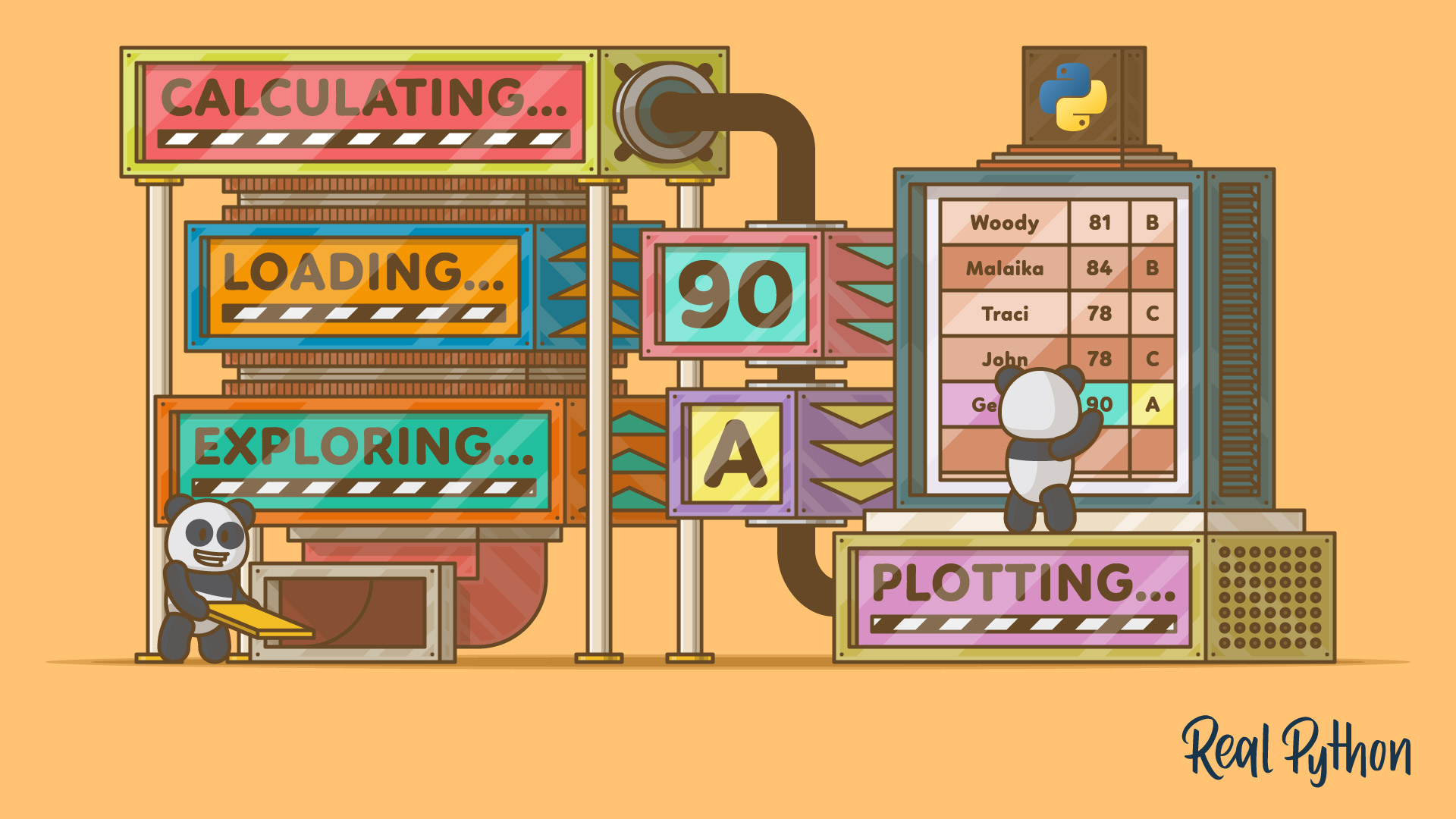
|  |  |
| --- | --- |
| **CRITERIA** | **MARKS** |
| CONTENTS | /6 |
| ORGANIZATION/STRUCTURE | /2 |
| WRITING MECHANICS | /2 |
| **TOTAL** | **/10** |

**FINAL REPORT**

In this section, students must display originality of their writings by jotting down their understanding of crawling. Students need to fully describe and discuss in terms of:

1. **The problem analysis and flowcharts for your python codes**
2. **Screenshot your running results (better applying video clips), upload your source codes and your final report.**

**Any of following topics you can select (Just select ONE of them):**



**Topic Two:**

Using Labwork 2' comments data from Amazon.

1. Analysis the part of speech for the review comments.
2. Display high frequency words with their 'pos' and counts
3. Display a wordcloud of entire comments.

**Tropic Two**

Finally, we are at the end of the course. This is out final work. So, I have tried my best to solve the problems. In this report there was 3 tasks, I choose the second one.

The task is about crawling amazon products comments and analysis the comments with parts of speech. Web crawling is the process of using bots to extract content and data from a website. Unlike screen scraping, which only copies pixels displayed onscreen, web crawling extracts underlying HTML code and, with it, data stored in a database. The crawler can then replicate entire website content elsewhere. It is an interesting task to get a websites data by scraping. I have used some Python libraries like selenium, BeautifulSoup to find data from the web page. Ans WordCloud, matplotlib to draw charts.

* Selenium: Selenium is a web testing library. It is used to automate browser activities.
* BeautifulSoup: Beautiful Soup is a Python package for parsing HTML and XML documents. It creates parse trees that is helpful to extract the data easily.
* Pandas: Pandas is a library used for data manipulation and analysis. It is used to extract the data and store it in the desired format.

**web crawler**

A web crawler is a program that systematically fetches web pages. Web crawlers are also known as ants, automatic indexers, bots, spiders, Web18 robots, and worms. It can create an index of available Web pages is through an automatic Web page discovery and retrieval system. It is an application that “automatically traverses the Web's hypertext structure by retrieving a document, and recursively retrieving all documents that are referenced”. An application that automatically traverses the Web might work as

follows:

* A base set of known working hyperlinks is used as a starting point.
* Each of these hyperlinks is placed in a queue. The Web crawler will endeavour to retrieve each of the pages in this queue. Because the Web is a hyperlinked environment, each of these pages will in all likelihood have links to various other pages on various other Web servers.
* The Web crawler retrieves the first page in its queue and adds an entry for this page in the index.
* Then, it adds each of the hyperlinks that exist on that Web page to the queue.
* It repeats this process of retrieval and recording for each page in its queue.

Thus, for each page discovered, the list of pages that remain to be gathered will (ideally) grow, and so will the index. Indeed, it becomes evident that a Web crawler’s indexing process is one that builds upon itself: the more pages it retrieves, the more pages it discovers that need to be added to its index.

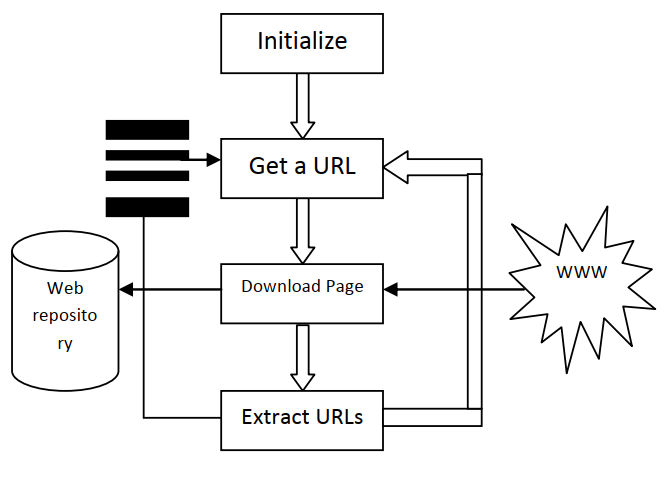


Figure 1: Flow of a crawling process

The working of a web crawler may be discussed as follows:

* Selecting a starting seed URL or URLs
* Adding it to the frontier
* Now picking the URL from the frontier
* Fetching the web-page corresponding to that URL
* Parsing that web-page to find new URL links
* Adding all the newly found URLs into the frontier
* Go to step 2 and reiterate till the frontier is empty

Thus, a web crawler will recursively keep on inserting newer URLs to the database repository of the search engine. So, we can see that the major function of a web crawler is to insert new links into the frontier and to choose a fresh URL from the frontier for further processing after every recursive step.

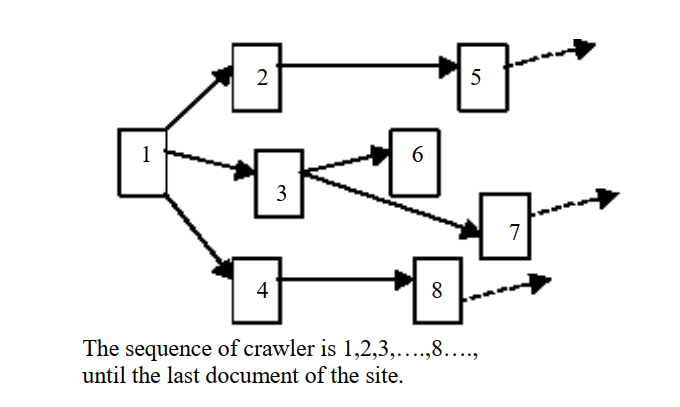


Figure 2: Web crawler example

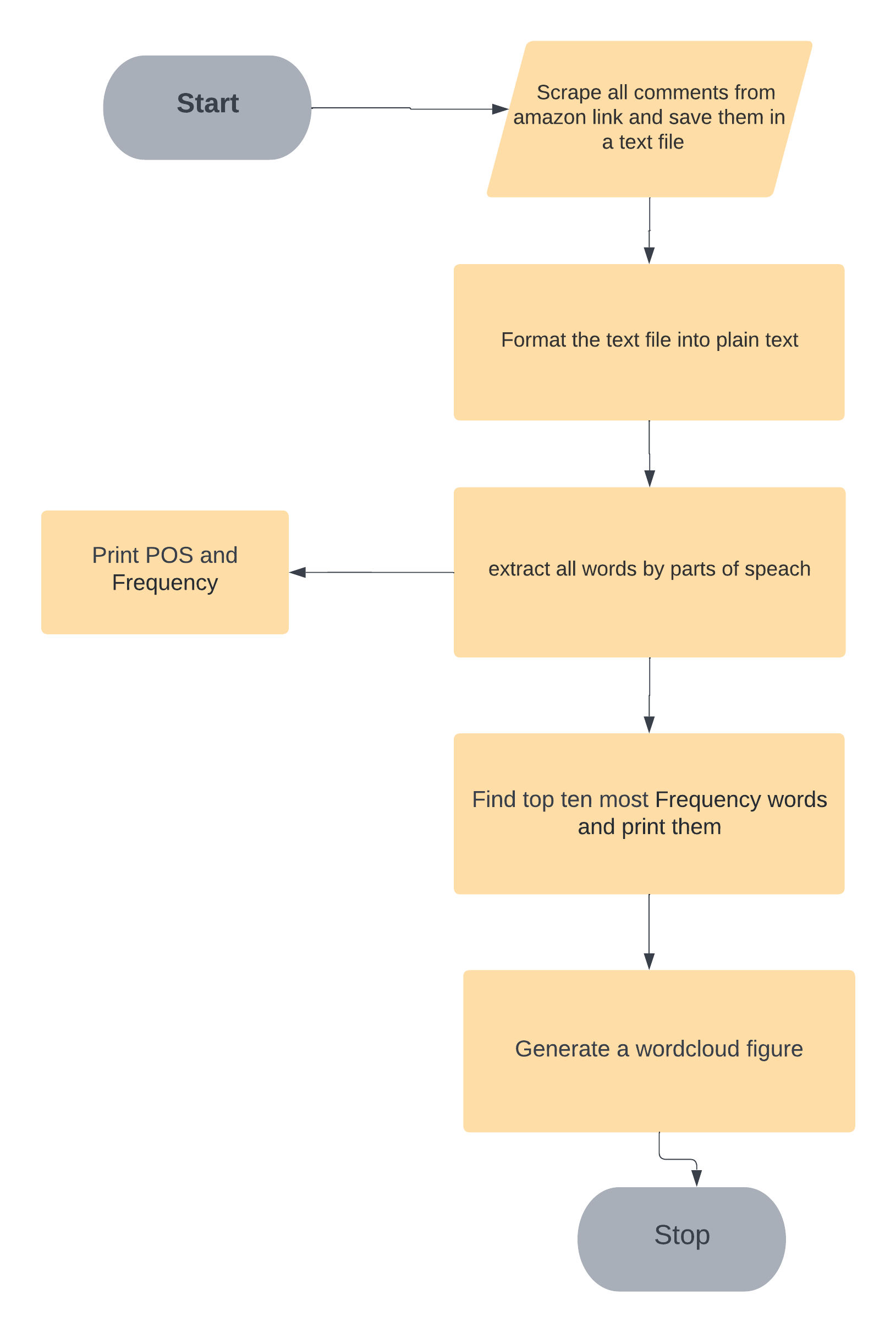
In this figure 2, a web crawler is discovering 7 pages from 1 initial page, moving left to right. The first three steps are depicted, the three dashed arrows at the end all point to further pages with more hyperlinks. This process of automatic page discovery takes a great burden off of a search engines maintenance team. Rather than having to rely on announcements and user-submissions to grow the index, the indexing application is constantly and automatically discovering new pages for inclusion in the index. Day-to-day maintenance of the index is aided greatly by the use of such an algorithm, but skilled programmers are still required to maintain and update the actual indexing application, as opposed to manual indexing where the human labor is focused on directly maintaining the index.

To be effective, the implementation of a Web-traversal algorithm needs to be done at a rather large scale. Even with an automated system such as this, indexing a significant portion of the Web is a gigantic task. Imagine

the indexing application were able to discover 1 million new pages a day (about 42,000 an hour or 700 per minute– possibly a generous estimate); it would take a bit over 2 years to index a static, non-changing body of

800,000,000 pages. Given that the turnover/change rate of the Web is significant–recall it is estimated to be around 40%. Many commercial search engines utilize variations on this approach to indexing. Such a demanding application also requires a fair amount of data transmission bandwidth simply for the indexing of Web pages. The implementation of a saleable and effective manual or automatic indexing scheme will almost certainly take on a large scale. There is an alternative: a scheme which makes no use of a persistent, local index.

**Flowchart**



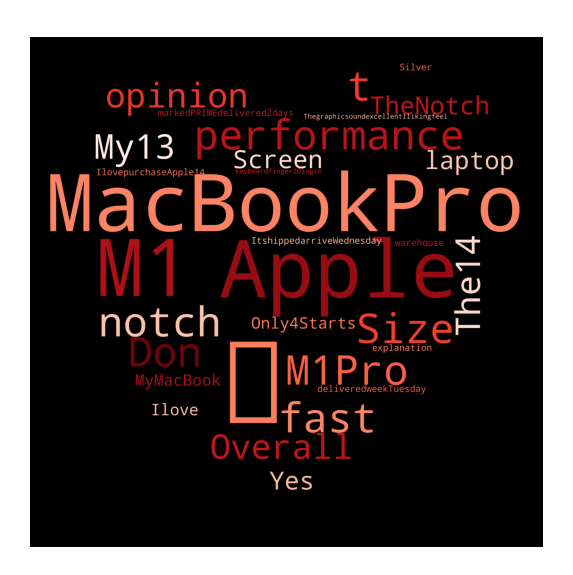
**Source code**

from selenium import webdriver  
from selenium.webdriver.common.by import By  
from time import sleep  
from bs4 import BeautifulSoup  
import pynlpir as pynlpir  
import pandas as pd  
import matplotlib.pyplot as plt  
from matplotlib.font\_manager import FontProperties  
import seaborn as sns; sns.set()  
from wordcloud import WordCloud  
from imageio import imread  
from collections import Counter  
  
  
web = webdriver.Chrome()  
web.maximize\_window()  
sleep(1)  
web.get('https://www.amazon.com/Apple-MacBook-14-inch-8%E2%80%91core-14%E2%80%91core/product-reviews/B09JQSLL92/ref=cm\_cr\_dp\_d\_show\_all\_btm?ie=UTF8&reviewerType=all\_reviews')  
  
sleep(2)  
page\_count = 0  
comment\_count = 0  
  
# --------------getting comments from the link--------------------  
  
file = open("comments.txt", "w")  
for i in range(13):  
 try:  
 sleep(3)  
 pageSource = web.page\_source  
 soup = BeautifulSoup(pageSource, 'html.parser')  
 reviews = soup.find('div', id="cm\_cr-review\_list").find\_all('div', class\_='a-section celwidget')  
  
 for review in reviews:  
 # collect all review contents  
 message = review.find('div', class\_='a-row a-spacing-small review-data').text  
 f\_message = message.strip()  
 file.write(f\_message)  
 # click next button  
 web.find\_element(by=By.CLASS\_NAME, value="a-last").click()  
  
 except:  
 continue  
file.close()  
web.close()  
  
# -------------- End getting comments from the link--------------------  
  
  
font = FontProperties(fname=r'c:\windows\fonts\arial.ttf', size=15)  
text\_file = open('comments.txt', 'r').read().replace('\n', '')  
  
pynlpir.open()  
pynlpir.segment(text\_file, pos\_names='parent', pos\_english=True)  
  
word\_list = []  
y\_list = []  
  
y\_list.extend(pynlpir.segment(text\_file, pos\_names='parent', pos\_english=True))  
  
for i in range(len(y\_list)):  
 y\_w = list(y\_list[i])  
 word\_list.append(y\_w)  
  
f\_words = pd.DataFrame(word\_list, columns=["word", "pos"])  
f\_words.head(25)  
f\_words.index.size  
  
stopword = open('stopwords\_en.txt', encoding='utf-8').read()  
  
for i in range(f\_words.shape[0]):  
 if f\_words.word[i] in stopword:  
 f\_words.drop(i, inplace=True)  
 else:  
 pass  
  
  
wordFormat = pd.DataFrame(f\_words["pos"].value\_counts(ascending=False))  
wordFormat.rename(columns={'pos': 'Frequency'}, inplace=True)  
  
print("\nPart of speech analysis for the review comments\n\n")  
  
# counting POS  
print(f"POS {wordFormat.head()}")  
print("\n--------------------------------\n")  
  
# counting Frequency  
print(f"Word Frequency : {wordFormat['Frequency'].sum()}")  
print("\n--------------------------------\n")  
  
# counting percentage of POS  
wordFormat['percentage'] = wordFormat['Frequency'] / wordFormat['Frequency'].sum()  
  
print(wordFormat['percentage'])  
print("\n--------------------------------\n")  
  
  
  
  
# Extracting words from the file by splitting by " "  
wordsCounter = open("comments.txt").read().split()  
  
# Print the top 10 most frequently occurred words in the file, and provide the word frequency for each of them.  
word\_freq = Counter(wordsCounter)  
print("\n\n---------High Frequency words in comments--------\n")  
df\_wordCount = pd.DataFrame(word\_freq.most\_common(10))  
df\_wordCount.columns = ["Words", "Frequency"]  
print(df\_wordCount)  
  
# # draw wordcloud  
myText = ''.join(f\_words.word)  
myText[:20]  
bg\_pic = imread('love.png')  
wc = WordCloud(mask=bg\_pic, max\_words=500, max\_font\_size=50, min\_font\_size=3, background\_color='black', colormap='Reds\_r', scale=15.5, contour\_color='red', repeat=True)  
  
wc.generate(myText)  
plt.imshow(wc)  
plt.axis('off')  
plt.show()  
pynlpir.close()

**Result**

"C:\Program Files\Python310\python.exe" "G:/1. SWPU/6th Semester/Data Science/pythonProject/data\_science/Final/FinalExam.py"  
  
Part of speech analysis for the review comments  
  
  
POS Frequency  
noun 3081  
punctuation mark 1069  
adjective 509  
verb 373  
numeral 205  
  
--------------------------------  
  
Word Frequency : 5368  
  
--------------------------------  
  
noun 0.573957  
punctuation mark 0.199143  
adjective 0.094821  
verb 0.069486  
numeral 0.038189  
adverb 0.016766  
multiword expression 0.005775  
preposition 0.001490  
time word 0.000186  
suffix 0.000186  
Name: percentage, dtype: float64  
  
--------------------------------  
  
  
  
---------High Frequency words in comments--------  
  
 Words Frequency  
0 the 427  
1 and 261  
2 a 234  
3 to 217  
4 I 206  
5 is 177  
6 of 146  
7 it 124  
8 for 113  
9 with 90  
  
Process finished with exit code 0

**Wordcloud**



**Reference**

1. <https://research.aimultiple.com/web-crawler/>
2. <https://www.scrapingbee.com/blog/web-scraping-101-with-python/>
3. <https://www.tutorialspoint.com/python_web_scraping/index.htm>
4. <https://www.geeksforgeeks.org/generating-word-cloud-python/#:~:text=Word%20Cloud%20is%20a%20data,highlighted%20using%20a%20word%20cloud>.
5. <https://www.analyticsvidhya.com/blog/2021/05/how-to-build-word-cloud-in-python/>
6. <https://www.geeksforgeeks.org/part-speech-tagging-stop-words-using-nltk-python/>
7. <https://www.dataknowsall.com/pos.html>
8. MetaCrawler Search Engine, available at: http://www.metacrawler.com.