VISVESVARAYA TECHNOLOGICAL UNIVERSITY
BELGAUM-590014

MINI-PROJECT ENTITLED

"WEB CRAWLER AND

SCRAPER IN PYTHON"

For the academic year 2019-2020
Submitted by:

AYUSHI	**1MV17CS022**
HARSH GAHLOT	**1MV17CS038**

Project carried out at
Sir M. Visvesvaraya Institute of Technology
Bangalore-562157.

Under the Guidance of

Mrs. SUPRIYA H.S
Assistant Professor,
Dept. of CSE
Sir MVIT, Bangalore.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
Sir M. Visvesvaraya Institute of Technology
HUNASAMARANAHALLI, Bangalore-562157.

CERTIFICATE

It is certified that the work contained in the project titled **"*Web Crawler and Scraper in python*"**, by **Ayushi (1MV17CS022) and Harsh Gahlot (1MV17CS038),** has been carried out under my supervision and that this work has done for Python Programming subject during the academic year 2019-2020. The project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the course of Bachelor of Engineering.

Name & Signature of Guide
Mrs. Supriya H.S
Assistant Prof. & Internal Guide
Dept. Of CSE, Sir MVIT
Bengaluru - 562157

Name & Signature of HOD
Dr. Bhanu Prakash G C
HOD, Dept. of CSE
Sir MVIT
Bengaluru -562157

ACKNOWLWDGEMENT

It gives us immense pleasure to express our sincere gratitude to the management of **Sir M. Visvesvaraya Institute of Technology, Bangalore** for providing the opportunity and the resources to accomplish our project work in their premises.

On the path of learning, the presence of an experienced guide is indispensable and we would like to thank our guide **Supriya H.S., Assistant Professor, Dept. of CSE,** for her invaluable help and guidance.

We would also like to convey our regards and sincere thanks to Dr. **Bhanu Prakash, HOD, Dept. of CSE** for his suggestions, constant support and encouragement, Heartfelt and sincere thanks to **Dr. V.R. Manjunath, Principal, Sir. MVIT** for providing us with the infrastructure and facilities needed to develop our project.

We would also like to thank the staff of Department of Computer Science and Engineering and lab-in-charges for their co-operation and suggestions. Finally, we would like to thank all our friends for their help and suggestions without which completing this project would not have been possible.

- Harsh Gahlot (1MV17CS038)
- Ayushi (1MV17CS022)

DECLARATION

We hereby declare that the entire mini project work embodied in this dissertation has been carried out by us and no part has been submitted for any degree or diploma of any institution previously.

Place: Bengaluru

Date: May 30, 2020

Signature of students:

HARSH GAHLOT (1MV17CS038)

AYUSHI (1MV17CS022)

ABSTRACT

This project aims at developing a highly efficient **WEB CRAWLER & SCRAPER** that browses the World Wide Web in a methodical, automated manner and can scrape the crawled data. A Web crawler is a computer program that browses the World Wide Web in a methodical, automated manner. Other terms for Web crawlers are ants, automatic indexers, bots, and worms or Web spider, Web robot, or especially in the FOAF community—Web scutter. The process is called **Web crawling or spidering**. **Web scraping** is the process of extracting data from websites. Some data that is available on the web is presented in a format that makes it easier to collect and use it, for example in the form of downloadable **comma-separated values (CSV)** datasets that can then be imported in a spreadsheet or loaded into a data analysis script and **pdf format**. Many sites, in particular search engines, use spidering as a means of providing up-to-date data. Web Crawlers are mainly used to create a copy of all the visited pages for later processing by a search engine that will index the downloaded pages to provide fast searches and then the scraper downloads the data in the pdf format.

CONTENTS

CHAPTER-1

INTRODUCTION

1.1 INTRODUCTION TO WEB CRAWLER AND SCRAPER

The terms Web Crawling and Scraping are often used interchangeably as the basic concept of them is to extract data. However, they are different from each other. We can understand the basic difference from their definitions.

Web crawling is basically used to index the information on the page using bots as known as crawlers. It is also called indexing. On the hand, web scraping is an automated way of extracting the information using bots as known as scrapers. It is also called **data extraction**.

A Web crawler and Scraper starts with a list of URLs to visit, called the *seeds*. As the crawler visits these URLs, it identifies all the hyperlinks in the pages and adds them to the list of URLs to visit, called the crawl frontier. URLs from the frontier are recursively visited according to a set of policies. If the crawler is performing archiving of websites, it copies and saves the information as it goes. The archives are usually stored in such a way they can be viewed, read and navigated as they were on the live web, but are preserved as ‘snapshots'.

The archive is known as the *repository* and is designed to store and manage the collection of web pages. The repository only stores HTML pages and these pages are stored as distinct files. A repository is similar to any other system that stores data, like a modern day database. The only difference is that a repository does not need all the functionality offered by a database system. The repository stores the most recent version of the web page retrieved by the crawler.

The large volume implies the crawler can only download a limited number of the Web pages within a given time, so it needs to prioritize its downloads. The high rate of change can imply the pages might have already been updated or even deleted.

The number of possible URLs crawled being generated by server-side software has also made it difficult for web crawlers to avoid retrieving duplicate content. Endless combinations of HTTP GET (URL-based) parameters exist, of which only a small selection will actually return unique content. For example, a simple online photo gallery may offer three options to users, as specified through HTTP GET parameters in the URL.

If there exist four ways to sort images, three choices of thumbnail size, two file formats, and an option to disable user-provided content, then the same set of content can be accessed with 48 different URLs, all of which may be linked on the site. This mathematical combination creates a problem for crawlers, as they must sort through endless combinations of relatively minor scripted changes in order to retrieve unique content.

1.2 MOTIVATION

To retrieve information from web, one has to know all relevant pages. Obviously, since world web contains billions of pages, it cannot be done manually. The process by which such data is gathered automatically is called web crawling and the program a web crawler, sometimes referred to as a spider. The goal of the web crawler is to gather as many web pages as possible efficiently. Crawler extracts links to other pages and other relevant data such as text, images, or server log files, every time it downloads (visits) a page. Then, it may visit one of the found pages and the process may be repeated, discovering a structure of the web. Web crawling is thus a process of graph exploration and collecting data. Information which was gathered may be used then to, e.g.,

- build a search engine (Google, etc.),
- analyse users behaviour (e.g., which pages are visited frequently, how users move through the web?),
- business intelligence (what are the current actions of my opponents or partners?),
- collecting e-mail addresses (phishing, sending spam).

A good crawler should provide the following features:

- ✓ **Freshness:** Pages (and web topology) may change over time. Some pages, such as news sites (e.g., BBC, CNN), change very frequently. In order to maintain up-to-date data, pages have to be visited again, from time to time. The more frequently a page changes the more often it should be visited by the crawler.
- ✓ **Quality:** The crawler should visit the most relevant and useful pages firstly. For instance, these pages that are frequently visited by users, updated, well maintained, and relevant to the underlying task (e.g., finding sports news only) should have high priority when determining next page to visit.

- ✓ **Politeness**: The crawler should respect web servers policies. These policies regulate which pages cannot or can, and how frequently, be visited by the crawler.
- ✓ **Robustness:** Since links among web pages turn the web into a directed graph, there are many so-called spider traps that are potential threats to web crawlers. These can be, e.g., graph cycles or dynamic pages.
- ✓ **Extensibility:** World web evolves over time. For instance, new file formats or standards may be developed. Good crawler should be based on modular architecture such that its functionality can be easily extended.
- ✓ **Efficiency:** System resources such as processors, memory, or network, should be used wisely to derive and store page information efficiently.
- ✓ **Distribution:** In case when many pages are to be crawled, this process needs to be distributed over multiple machines allowing to retrieving data from multiple pages at the same time.
- ✓ **Scalability:** Crawling rate should be easily improved by adding new resources, e.g., processors or memory.

1.3 AIM OF THIS PROJECT

This project aims at developing a highly efficient **WEB CRAWLER & SCRAPER** that browses the World Wide Web in a methodical, automated manner. A Web crawler is a computer program that browses the World Wide Web in a methodical, automated manner. Other terms for Web crawlers are ants, automatic indexers, bots, and worms or Web spider, Web robot, or especially in the FOAF community—Web scutter. The process is called **Web crawling or spidering**. Many sites, in particular search engines, use spidering as a means of providing up-to-date data. Web Crawlers are mainly used to create a copy of all the visited pages for later processing by a search engine that will index the downloaded pages to provide fast searches.

CHAPTER-2

LITERATURE REVIEW

2.1 HISTORY

The history of the web scraping dates back nearly to the time when the Internet was born.

- ✓ After the birth of World Wide Web in 1989, the first web robot, World Wide Web Wanderer, was created in June 1993, which was intended only to measure the size of the web.
- ✓ In December 1993, the first **crawler-based web search engine**, JumpStation, was launched. As there were not so many websites available on the web, search engines at that time used to rely on their human website administrators to collect and edit the links into a particular format. In comparison, JumpStation brought a new leap, being the first WWW search engine that relied on a web robot.
- ✓ RBSE spider developed and used by the NASA funded Repository Based Software Engineering (RBSE) program in the year 1994, at the University of Houston, Clear Lake. It was built by David Eichmann of NASA using the languages Oracle, C, and wais. The primary purpose of this crawler was indexing and statistics source. At the time when this crawler was built, size of the web was just about 100,000 web pages.
- ✓ WebCrawler created by Brian Pinkerton of the University of Washington and launched on April 20, 1994, WebCrawler was the first search engine that was powered by a web crawler. According to Wikipedia, WebCrawler was the first web search engine to provide full text search.
- ✓ In 2000, the **first Web API and API crawler** came. API stands for **Application Programming Interface**. It is an interface that makes it much easier to develop a program by providing the building blocks. In 2000, Salesforce and eBay launched their own API, with which programmers were enabled to access and download some of the data available to the public. Since then, many websites offer web APIs for people to access their public database.

2.2 EXISTING SYSTEM

The content in the web is not managed by a single person but consists of millions of people managing their respective content. During the beginnings of the Internet, There were two models for content aggregation. One was the Pull model, where the search engines and other content aggregators employ a process which actively checks for new and updated pages.

The other was the Push model where they could enable the content providers to push content to them. But the Push model eventually faded out and Pull model became almost the only model of web crawling. Push model did not succeed because it raised the barrier of entry into the web as the content providers had to follow protocols for pushing information to the aggregators and some trust had to be established between the provider and aggregator before an aggregator accepts to include its information in the search index. Whereas in the pull model, the provider just needs to put his content in the server and need not care much about whether the search engine is showing his website in the search results, unless he wants to prevent some crawlers from accessing certain parts of the website. For this purpose a protocol called robots exclusion protocol has been developed and the crawlers are expected to follow the guidelines provided in it.

The Pull model has gradually evolved into what we now call as Web crawling. The basic architecture of web crawling appears very simple, but there are many optimizations that should be done to the algorithms, data structures and also the hardware that are used.

2.3 PROPOSED SYSTEM

To build a web crawler, one must-do step is to **download the web pages**. This is not easy since many factors need to be taken into consideration, like how to better leverage the local bandwidth, how to optimize DNS queries, and how to release the traffic in the server by assigning web requests reasonably.

After we fetch the web pages, **the HTML pages complexity analysis** follows. In fact, we can’t get all HTML web pages straight. And here comes another issue. How to retrieve the content generated by JavaScript when AJAX is used everywhere for dynamic websites? Plus, the Spider Trap occurring frequently on the Internet would make an infinite number of requests or cause a poorly constructed crawler to crash.

While there are many things we should be aware of when building a web crawler, in most cases we just want to create a crawler for a specific website. Thus, we'd better do deep research on the structure of target websites and pick up some valuable links to keep track of, in order to prevent extra cost on redundant or junk URLs.

CHAPTER-3

SPECIFICATIONS

3.1 HARDWARE REQUIREMENTS

- PROCESSOR : Intel Pentium or Higher Version
- RAM : Minimum 1GB
- HARD DISK : 60GB and above

3.2 SOFTWARE REQUIREMENTS

- SOFTWARE : Python 3.3 or greater
- SUPPORTED BROWSERS : Google Chrome / Mozilla Firefox / Internet Explorer
- EDITOR : Atom / Visual Studio Code
- FRAMEWORK : beautifulsoup4 4.9.0
- OPERATING SYSTEM : Windows , or MacOS, or Linux (32/64 bit)
- PROGRAMMING LANGUAGE: python 3.6.0 and above versions.

3.3 FUNCTIONAL REQUIREMENTS

The Functional Requirements Specification documents the Operations and activities that a system must be able to perform. Functional Requirements include:

- To crawl the web pages at the base URL.
- To take the information from the user about the topics specified.
- To scrap all the pages that the user needs.
- Convert the scraped information into a ppt.

The Functional Requirements Specifications is designed to be read by a general audience. Readers should understand the system, but no particular technical knowledge should be required to understand the document. These are the functional requirements specification documents for the project analysis.

A software requirement specification helps to attenuate the time and energy needed by the developers to attain their desired goals and additionally minimizes the value of development.

Following Factors are used to measure software development quality:

Each attribute may be accustomed measure of the product performance. These attributes may be used for Quality assurance similarly as quality control. Quality assurance activities are directed towards prevention of introduction of defects and internal control activities are aimed toward detecting defects in product and services.

1. **Reliability**

   Measure if product is reliable enough to sustain in any condition. Give systematically correct results. Product dependability is measured in terms of operation of project underneath different operating atmosphere and different conditions.

2. **Maintainability**

   Different versions of the product ought to be easy to maintain. For development it ought to be easy to feature code to existing system, ought to be easy to upgrade for brand new options and new technologies time to time. Maintenance ought to be value effective and simple. System be easy to take care of and correcting defects or making a change within the software system.

3. **Usability**

   This can be measured in terms of ease of use. Application should be user friendly. Easy to use for input preparation, operation and also for interpreting of output.

4. **Portability**

   This can be measured in terms of Costing issues related to porting, Technical issues related to porting, Behavioural issues related to porting.

3.4 NON-FUNCTIONAL REQUIREMENTS

Satisfactory will probably not be assessed on the system where the program is developed, tested or first installed.

CHAPTER-4

SYSTEM DESIGN

4.1 INTRODUCTION

System design is the first design stage for devising the basic approach to solving the problem. During system design, developers decide the overall structures and styles. The system architecture determines the organization of the system into subsystems. In addition, the architecture provides the context for the detailed decisions that are made in later stages. during design, developers make decisions about how the problem will be solved, first at the high level and then with more detail.

4.2 ARCHITECTURAL DIAGRAM

A **architectural diagram** represents the conceptual model that defines the structure, behaviour, and more views of a system. An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structures and behaviors of the system.

A system architecture can consist of system components and the sub-systems developed, that will work together to implement the overall system. There have been efforts to formalize languages to describe system architecture, collectively these are called architecture description languages (ADLs).

4.3 DATA FLOW DIAGRAM

A Data Flow Diagram (DFD) is a graphical representation of the “flow” of data through an information system. Data Flow modules are used to show how data flows through a sequence of processing steps. The data is transformed at each step before moving on to the next stage. These processing steps or transformations are program functions when Data Flow Diagrams are used to document a software design.

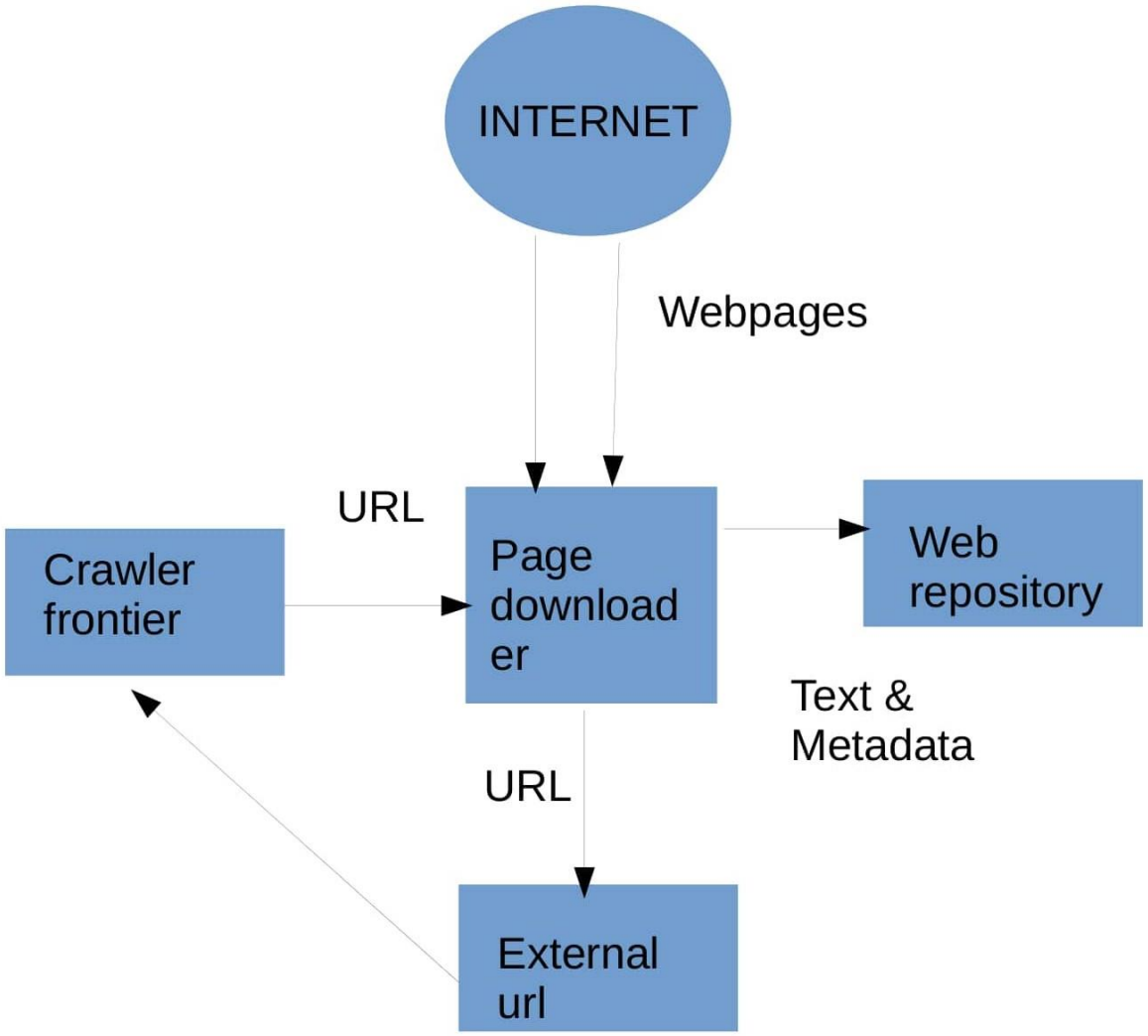
Data flow modules are an intuitive way of showing how data is processed by a system. At the analysis level, they should be used to module the way in which data is processed in the existing system. The notation used in these modules represents functional processing, data stores and data movements between functions.

With a dataflow diagram, users are able to visualize how the system will operate, what the system will accomplish and how the system will be implemented. Old system dataflow diagrams can be drawn up and compared with the new system dataflow.

These are several common modelling rules to be followed while creating DFD's are as follows:

- All processes must have at least one data flow in and one data flow out.
- All processes should modify the incoming data, producing a new form of outgoing data.
- Each data store must be involved with at least one data flow.
- Each external entity must be involved with at least one data flow.
- A data flow must be attached to at least one process.

ARCHITECTURAL DIAGRAM



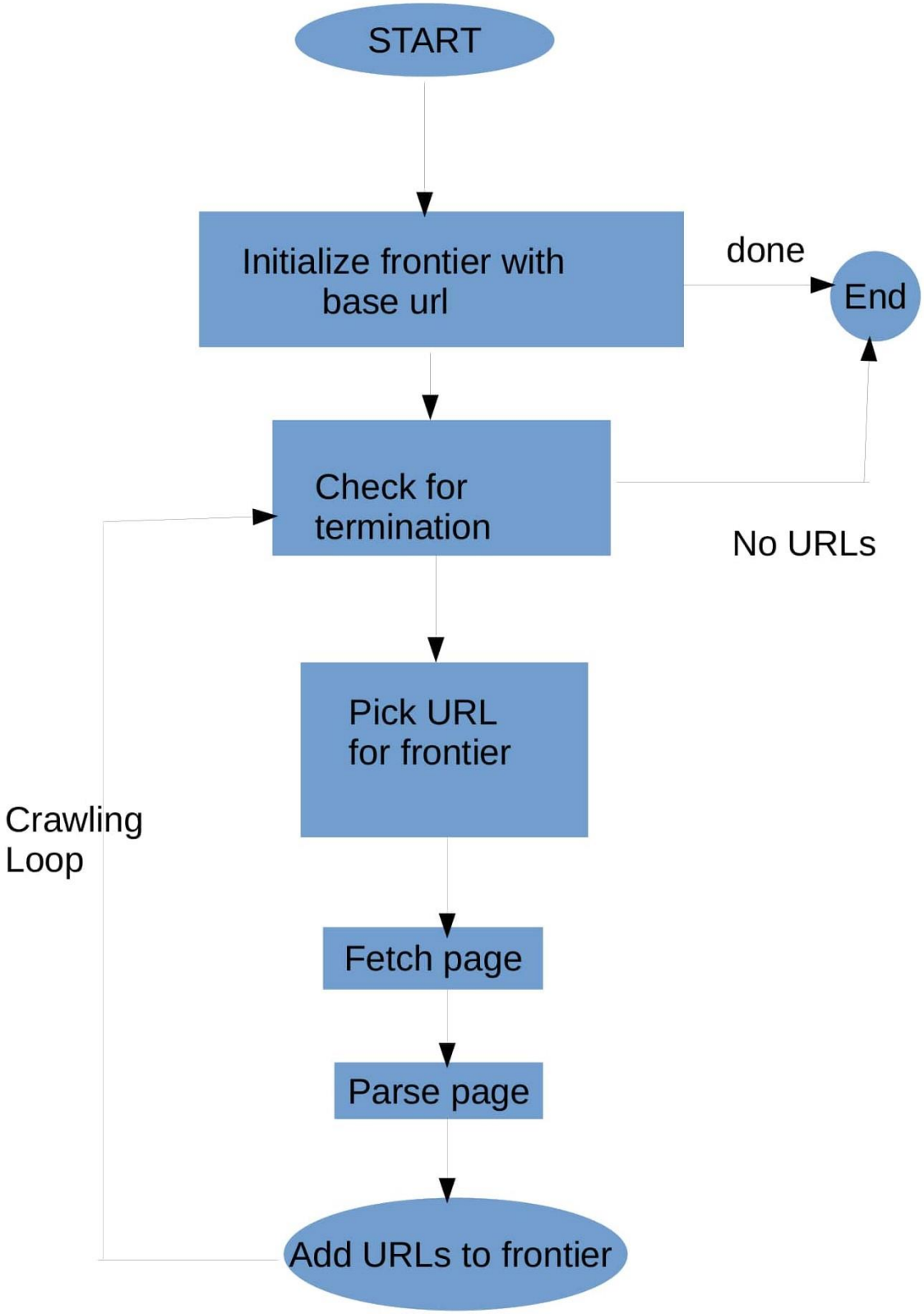
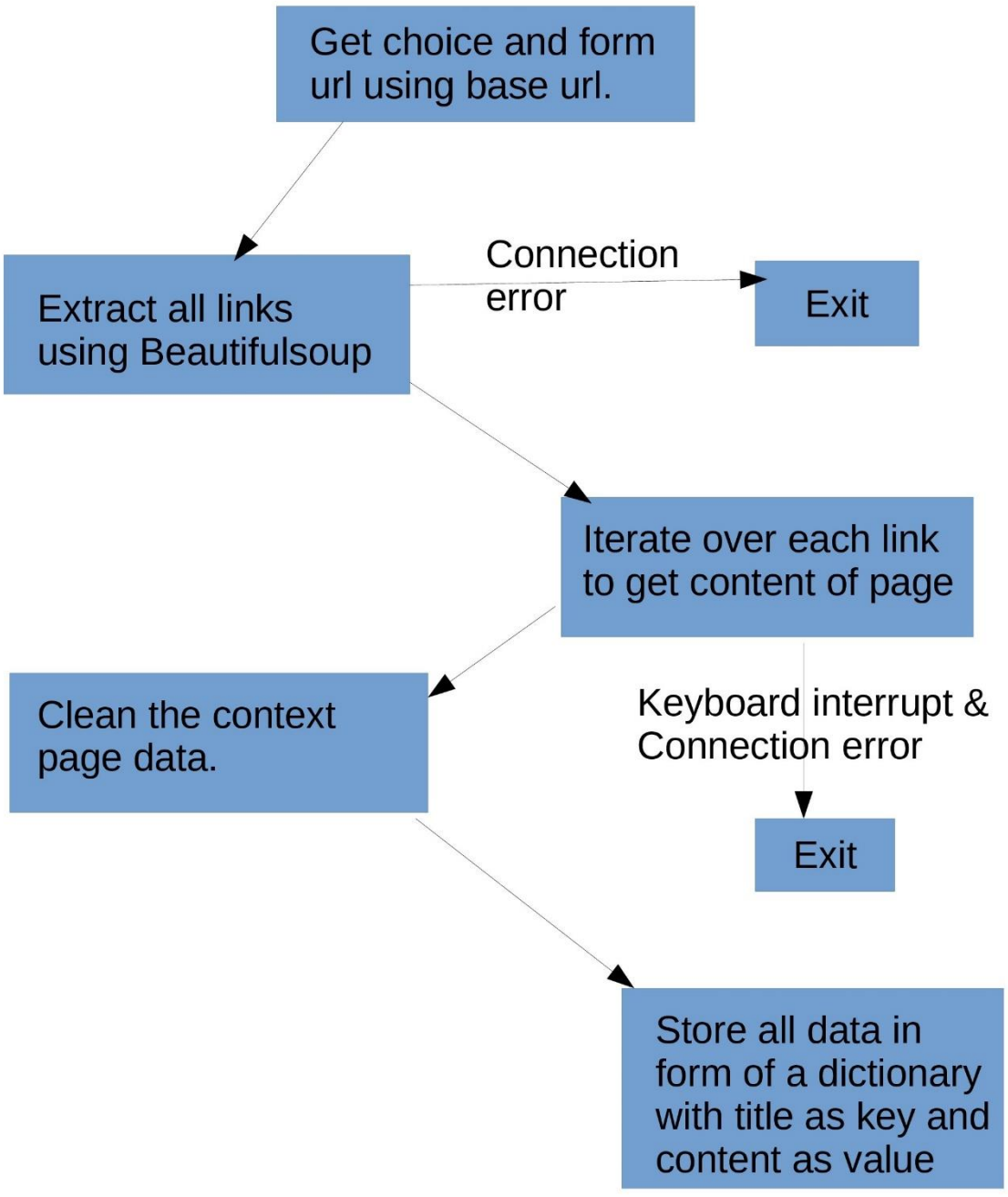


Fig 4.1 Architectural Diagram

DATA FLOW DIAGRAM



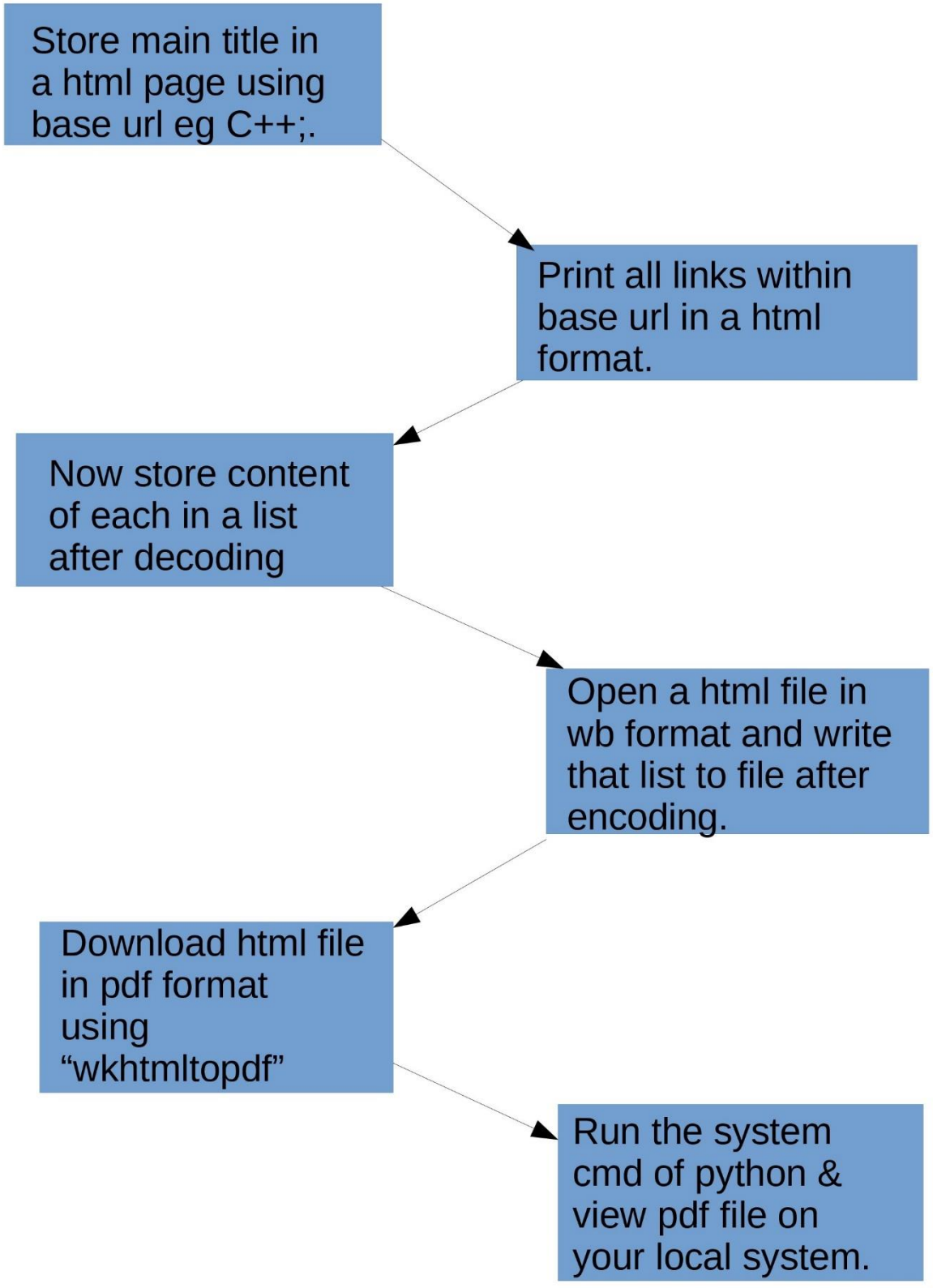


Fig 4.2 Data Flow Diagram

CHAPTER-5

PROJECT DEMONSTRATION AND TESTING

5.1 MODULE COVERAGE

In the first stage of implementation we will load all the pre-requisite modules by importing them in the program file. The program file will be of .py extension which means it is a python file.

Following pythons modules will be used here:

1. **For sending HTTP requests**:
   The requests module allows you to send HTTP requests using Python.

```
import requests
```

2. **For interacting with the OS:** The OS module in python provides functions for interacting with the operating system. OS, comes under Python's standard utility modules. This module provides a portable way of using operating system dependent functionality.

```
from os import system
```

3. **For exiting the function:**
   The exit function in python's sys module is used to exit any function.

```
from sys import exit
```

4. **For the program to wait:** The time.sleep pauses execution, making the program wait for the defined time. The time to wait (sleep) is defined in seconds**.**

```
from time import sleep
```

5. **For a Connection error occurred:**

```
from requests.exceptions import ConnectionError
```

6. **For scraping data:** Beautiful Soup is a Python library for pulling data out of HTML and XML files. It works with your favourite parser to provide idiomatic ways of navigating, searching, and modifying the parse tree.

```
from bs4 import BeautifulSoup
```

5.2 METHODOLOGY

1. **Store base URL as:**

```
BASE_URL = "https://www.geeksforgeeks.org/"
```

2. **Scrape the contents of base page and linked pages:**

```
try:
    soup = BeautifulSoup(requests.get(BASE_URL + category_url).text,"lxml")
except ConnectionError:
    print("Couldn't connect to Internet! Please check your connection & Try again.")
    exit(1)
    links = [a.attrs.get("href") for a in soup.select("article li a")]
    link_soup = BeautifulSoup(requests.get(link).text)
```

3. **Clean the web pages and store them in encoded form:**

```
[script.extract() for script in link_soup(["script", "ins"])]
for code_tag in link_soup.find_all("pre"):
    code_tag["class"] = code_tag.get("class", []) + ["prettyprint"]
    article = link_soup.find("article")
    page = Article(title=link_soup.title.string, content=article.encode("UTF-8"))
    articles.append(page)
```

4. **Form initial of web page:**

```
all_articles = (
        "<!DOCTYPE html>"
        "<html><head>"
        '<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />'
        '<link rel="stylesheet" href="style.min.css" type="text/css" media="all" />'
        '<script src="https://cdn.rawgit.com/google/code-
prettify/master/loader/run_prettify.js"></script>'
        "</head><body>"
         )
all_articles += ('<h1 style="text-align:center;font-size:40px">' + category_url.title()
          + " Archive</h1><hr>")
all_articles += '<h1 style="padding-left:5%;font-size:200%;">Index</h1><br/>'

for x in range(len(articles)):
    all_articles += ('<a href ="#' + str(x + 1) + '">' + '<h1 style="padding-left:5%;
                    font-size:20px;"> + str(x + 1) + ".\t\t" + articles[x].title
                    + "</h1></a> <br/> )
```

5. **Get content from respective links and write to html page opened in wb format:**

```
all_articles += """</body></html>"""

    html_file_name = "G4G_" + category_url.title() + ".html"
    html_file = open(html_file_name, "wb")
    html_file.write(all_articles.encode("utf-8"))
    html_file.close()
```

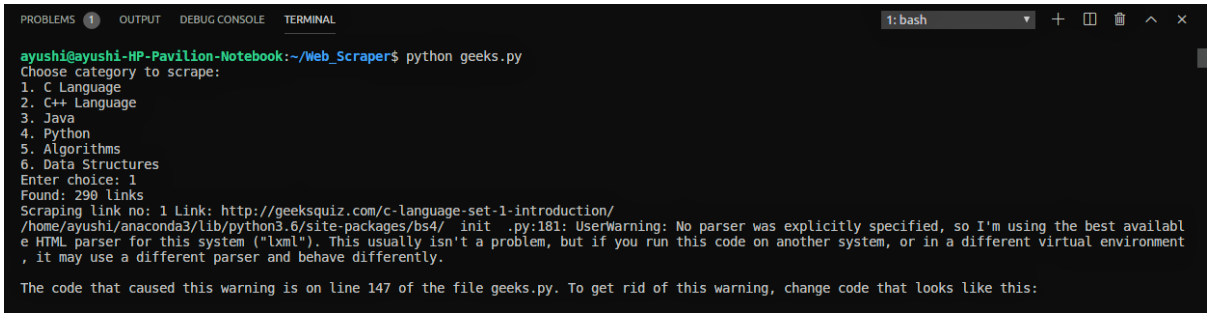
6. **Convert html page into pdf:**
   wkhtmltopdf is able to put several objects into the output file, an object is either a single webpage, a cover webpage or a table of contents. The objects are put into the output document in the order they are specified on the command line, options can be specified on a per object basis or in the global options area. Options from the Global Options section can only be placed in the global options area.

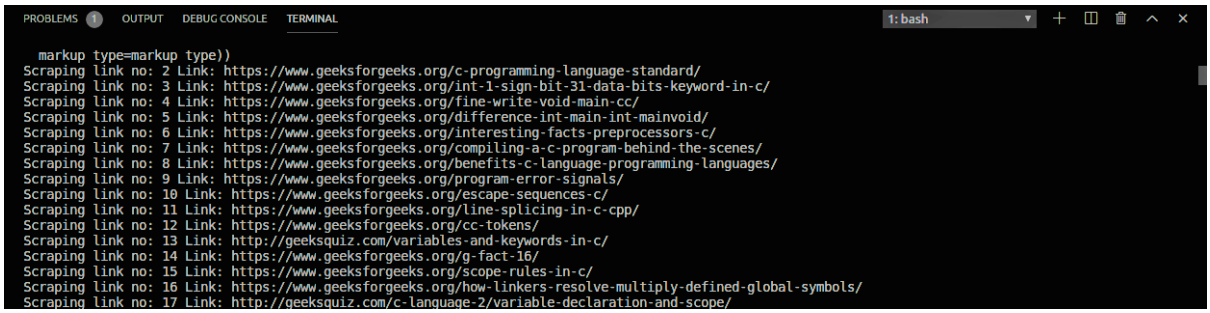
```
pdf_file_name = "G4G_" + category_url.title() + ".pdf"
print("Generating PDF " + pdf_file_name)
html_to_pdf_command = "wkhtmltopdf " + html_file_name + " "  +
                                pdf_file_name
system(html_to_pdf_command)
```

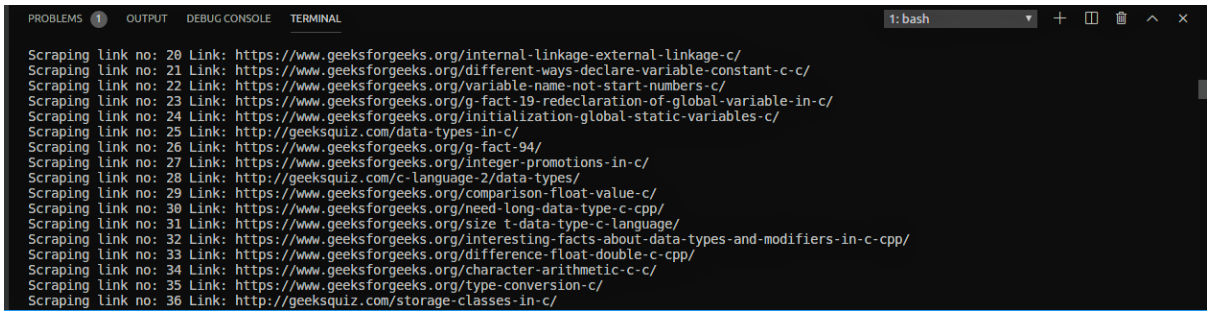
CHAPTER-6

RESULTS AND SNAPSHOT

6.1 PROGRAM OUTPUT







6.2 DOWNLOADED OUTPUT FILE

CONCLUSION

Web Crawler is the vital source of information retrieval which traverses the web and downloads web documents that suits the user's need. Web crawler is used by the search engine and other users to regularly ensure that their database is up-to-date. Web crawlers are a central part of search engines, and details on their algorithms and architecture are kept as business secrets. When crawler designs are published, there is often an important lack of detail that prevents others from reproducing the work. To get automatic information from website, web scraping is the most effective and efficient technique. Among all other techniques mention in this paper which are used to extract and store data, web scraping is more reliable, fast and automatic data retrieval system. By using web scraping terminology user can easily extract unstructured data on single or multiple websites into a structured data automatically.

To sum up, web crawlers play a huge role in the Internet era. Without web crawlers, you can't imagine how difficult it is to find the information you want among such an information ocean.

REFERENCES

[1] https://en.wikipedia.org/wiki/Web_crawler

[2] https://en.wikipedia.org/wiki/Web_scraping

[3] https://researchdata.wisc.edu/news/an-introduction-to-web-scraping-for-research/

[4] http://www.cs.put.poznan.pl/alabijak/ezi/lab1/Lab1-Crawling-Python

[5] S.C.M. de S Sirisuriya,2015, A Comparative Study on Web Scraping .Proceedings of 8th International Research Conference, KDU.