**REPORT:**

**Web Scraping Script to Extract Main**

**Headings from an article**

**SUBMITTED BY: ARBAZ AHMAD**

**INTRODUCTION**

Web Scraping, also known as web harvesting or web data extraction, is used to extract data from websites. Although it can be done manually by a user, it is generally done by a bot or an AI controlled web crawler. In this data extraction information of similar types is extracted from a web site and is stored in a spreadsheet or a central local database, which is later used for analysis. Webpages are build using text-based mark-up language, which generally contains useful data in text format. However, as we see these pages do not display the HTML directly and are made for the end user and not made for automated use. As a result, the web scrapers come into play and make our lives easier.

The purpose of this report is to describe a Python script that utilizes web scraping techniques to extract main headings from a specified article URL. The script uses the ‘**requests’** library to fetch the web page content and the ‘**BeautifulSoup**’ library for parsing and navigating the HTML content. The extracted main headings are then printed to the console.

**OBJECTIVE**

The primary objective of this project is to demonstrate the development of a web scraping script capable of efficiently extracting main headings from a web page. Main headings, often represented by h2 tags in HTML, are crucial elements that highlight the core topics discussed within an article or web page. By focusing on extracting headings, this project aims to showcase a specific use case for web scraping that serves as a foundation for more complex data extraction tasks. The report also seeks to create awareness regarding the ethical considerations in web scraping. Respecting website terms of service, obtaining proper permissions, and adhering to ethical practices are essential to ensure that web scraping remains a responsible and sustainable data retrieval approach.

**BACKGROUND**

The exponential growth of the internet has led to an information explosion, making data retrieval a fundamental aspect of modern-day research and decision-making. Web scraping, as a technique to automate data extraction from web pages, has become increasingly significant in various domains. Researchers can gather vast datasets for analysis, businesses can monitor competitors and market trends, and analysts can gain insights for strategic planning.

However, web scraping is a double-edged sword, as misuse and unethical practices can lead to legal issues and disrupt the equilibrium of the web ecosystem. Therefore, it is crucial to understand the ethical implications and adhere to website policies while performing web scraping.

**HARDWARE AND SOFTWARE REQUIREMENTS**

The successful execution of the web scraping script relies on the following hardware and software requirements:

**-Hardware Requirements:**

- A computer with sufficient processing power and memory to run the Python script smoothly.

- A stable internet connection to send HTTP requests and receive responses from the web server.

**- Software Requirements:**

- Python (version 3.x recommended) installed on the system.

- The 'requests' library for sending HTTP requests to the web page.

- The 'BeautifulSoup' library for parsing the HTML content of the web page.

**CODING**

The web scraping script has been developed with a structured approach to ensure efficient data extraction. Here are the key steps of the script:

**-Step 1: Importing Necessary Libraries**

The script begins by importing the 'requests' and 'BeautifulSoup' libraries, which are fundamental to sending HTTP requests and parsing the HTML content, respectively.

import requests

from bs4 import BeautifulSoup

**-Step 2: Setting the Target URL**

The URL of the web page from which we want to extract the main headings is assigned to the variable 'article\_url'. In this case, we are using ‘https://www.bbc.com/news/world’

*# URL of the article you want to scrape*

article\_url = 'https://www.bbc.com/news/world'

**-Step 3: Sending HTTP Request and Fetching Content**

The script sends an HTTP GET request to the 'article\_url' using the 'requests.get()' method and stores the response in the 'response' variable.

*# Fetch the web page content*

response = requests.get(article\_url)

**-Step 4: Validating the Request**

To ensure that the request was successful, the script checks the status code of the response. A status code of 200 indicates a successful request, while other codes indicate errors or issues.

*# Check if the request was successful (status code 200)*

response.**status\_code** == 200

**-Step 5: Parsing HTML Content**

If the request was successful, the HTML content of the web page is parsed using BeautifulSoup. This enables us to navigate the HTML structure and extract specific elements like headings.

*# Parse the HTML content*

soup = BeautifulSoup(response.**text**, 'html.parser')

**-Step 6: Extracting and Printing Main Headings**

Using the 'soup.find\_all()' method, the script identifies all occurrences of main headings (h1, h2, h3, h4, h5, h6) present in the parsed HTML. The text of each heading is then extracted using the 'heading.get\_text().strip()' method and printed.

*# Find all main headings (h1, h2, h3, h4, h5, h6)*

main\_headings = soup.find\_all(['h1', 'h2', 'h3', 'h4', 'h5', 'h6'])

*# Extract the text from the headings and print them*

if main\_headings:

        print("Main Headings:")

        for heading in main\_headings:

            print(heading.get\_text().strip())

    else:

        print("No main headings found in the article.")

else:

    print("Failed to fetch the article. Status code:", response.**status\_code**)

**OUTPUT SCREENSHOT**



**FUTURE SCOPE**

While the current implementation successfully achieves its main objective of extracting main headings, there is significant potential for future enhancement and expansion of the project. Some areas that can be explored for further improvement include:

**- Error Handling:** Implementing comprehensive error handling to gracefully deal with various HTTP response codes and exceptions, ensuring a smoother user experience.

**- Metadata Extraction:** Expanding the script to extract additional metadata from the web page, such as the publication date, author information, and article summaries.

**- User Interface Development:** Creating a user-friendly interface that allows users to input custom URLs, choose specific elements for extraction, and visualize the results in a more organized format.

**CONCLUSION**

In conclusion, the web scraping project serves as an illustrative example of how Python and BeautifulSoup can be leveraged to extract main headings from a web page. Web scraping offers a powerful solution for automating data retrieval and enables researchers, businesses, and data analysts to access critical information for various applications. Nevertheless, it is essential to emphasize ethical web scraping practices. Respecting website terms of service, seeking proper permissions, and adopting responsible data extraction practices are paramount to maintaining a positive and collaborative web environment. With the ever-expanding digital landscape, web scraping continues to be a valuable and indispensable tool in the hands of responsible users, contributing to data-driven insights and informed decision-making.

**REFERENCES AND BIBLIOGRAPHY**

During the development of this project, the following resources were instrumental in gaining insights and knowledge:

- [Requests Documentation](https://docs.python-requests.org/en/latest/): Official documentation for the 'requests' library, providing comprehensive guidance on using the library to send HTTP requests.

- [BeautifulSoup Documentation](https://www.crummy.com/software/BeautifulSoup/bs4/doc/): The official documentation for BeautifulSoup, offering a detailed reference to navigate and parse HTML content effectively.

- [Python Official Website](https://www.python.org/): The official Python website, providing essential information on Python programming and its various libraries and modules.

- [Longreads](https://longreads.com/2023/07/20/barbenheimer-reading-list/): The longreads website, the source of the target web page used for demonstration purposes in this project.