**Scraping Web Page URLs**

Divolika Bajpai

June 8, 2024

# INTRODUCTION

Developing a Python programme that automates web scraping from Google search results is the aim of this assignment. For a given query, the programme will run a Google search, compile a list of URLs from the search results, and visit each URL to extract data from the web pages. The programme will gather and save the data in a document format by using libraries like Requests and BeautifulSoup for handling web requests and parsing HTML, and the Google Colab files module for file management. This automation highlights key web scraping and data extraction strategies while streamlining the process of gathering data from the internet.

# METHODOLOGY

## OBJECTIVE

The objective of this assignment is to write a Python program to:

1) Perform a Google search for a given query.
2) Extract URLs from the Google search results.
3) Open each extracted URL and scrape the data from the individual web pages.
4) Store the scraped data in a document format.

## TOOL and LIBRARIES used

1) Requests: To send HTTP requests to web pages.
2) BeautifulSoup: To parse HTML and extract data from web pages.
3) Urllib.parse: To parse URLs and handle query strings.
4) Google Colab files module: To handle file download in Google Colab.

## IMPLEMENTATION STEPS

1. Scraping Google Search Results: The scrape_google function sends a GET request to the Google search URL with the provided query and extracts URLs from the search results.

2. Scraping Individual Web Pages: The scrape_individual_page function sends a GET request to each individual URL, parses the HTML content using BeautifulSoup, and extracts the title and text content of the page.

3. Main Function to Integrate the Workflow

1. Scrapes the Google search results to extract URLs.
2. Scrapes each individual URL to gather the required data.
3. Stores the scraped data in a text file and downloads it.

```python
def scrape_google(query):
    # Google search URL
    url = f"https://www.google.com/search?q={query}"

    # Sending a GET request to the Google search URL
    response = requests.get(url)

    # Parsing the response content using BeautifulSoup
    soup = BeautifulSoup(response.text, 'html.parser')

    # Extracting all search result links
    search_results = soup.find_all('a', href=True)

    # Extracting URLs from search results
    urls = []
    for link in search_results:
        url = link['href']
        if url.startswith('/url?q='):
            urls.append(url[7:])

    return urls
```

Figure 1: scrape_google(query)

## EXPLANATION OF KEY FUNCTIONS

- scrape_google(query):

1) Constructs a Google search URL with the provided query.
2) Sends a GET request to Google.
3) Parses the HTML content using BeautifulSoup.
4) Extracts URLs from the search result links that start with /url?q=.

- scrape_individual_page(url):

```python
def scrape_individual_page(url):
    # Sending a GET request to the individual URL
    response = requests.get(url)

    # Parsing the response content using BeautifulSoup
    soup = BeautifulSoup(response.text, 'html.parser')

    # Scraping data from the individual page
    title_element = soup.find('title')
    if title_element:
        title = title_element.get_text()
    else:
        title = "No title found"

    paragraphs = soup.find_all('p')
    text_content = '\n'.join([p.get_text() for p in paragraphs])

    return {'title': title, 'content': text_content}
```

Figure 2: scrape_individual_page(url)

1) Sends a GET request to the provided URL.
2) Parses the HTML content using BeautifulSoup.
3) Extracts the title and text content (paragraphs) from the web page.

- main():

```
def main():
    query = input("Enter your search query: ")
    num_pages = int(input("Enter the number of search results/pages to scrape: "))

    all_data = []

    # Scrape Google search results
    for page_num in range(num_pages):
        urls = scrape_google(query)

        # Scrape individual pages
        for url in urls:
            parsed_url = urlparse(url)
            if parsed_url.netloc:
                data = scrape_individual_page(url)
                all_data.append(data)

    # Save the results to a document
    with open("scraped_results.txt", "w") as file:
        for data in all_data:
            file.write(f"Title: {data['title']}\n\n")
            file.write(f"Content: {data['content']}\n\n\n")

    # Download the document
    files.download("scraped_results.txt")

if __name__ == "__main__":
    main()
```

Figure 3: main()

1) Prompts the user for the search query and the number of search result pages to scrape.
2) Calls scrape_google to get a list of URLs from Google search results.
3) Iterates over the extracted URLs and calls scrape_individual_page to get the data.
4) Writes the scraped data to a text file and downloads it using Google Colab's file handling.

# CONCLUSION

In conclusion, this assignment demonstrated how to automate the process of web scraping using Python. By leveraging libraries like Requests and BeautifulSoup, we successfully implemented a program that performs Google searches, extracts URLs, and scrapes data from individual web pages. The scraped data was then compiled and saved into a document format, illustrating practical applications of web scraping and data extraction. This project not only simplifies data collection from the web but also provides a solid foundation for more advanced web scraping and data analysis tasks.